Dvips: A DVI-to-PostScript Translator

for version 5.58f
January 1995

Tomas Rokicki
(edited for Dvipsk by ‘kb@cs.umb.edu’)

1 Why use Dvips?

The Dvips program has a number of features that set it apart from other PostScript drivers for TeX. This rather long section describes the advantages of using dvips, and may be skipped if you are just interested in learning how to use the program. See Chapter 9 [Installation], page 50, for details of compilation and installation.

The dvips driver generates excellent, standard PostScript, that can be included in other documents as figures or printed through a variety of spoolers. The generated PostScript requires very little printer memory, so very complex documents with a lot of fonts can easily be printed even on PostScript printers without much memory, such as the original Apple LaserWriter. The PostScript output is also compact, requiring less disk space to store and making it feasible as a transfer format.

Even those documents that are too complex to print in their entirety on a particular printer can be printed, since dvips will automatically split such documents into pieces, reclaiming the printer memory between each piece.

The dvips program supports graphics in a natural way, allowing PostScript graphics to be included and automatically scaled and positioned in a variety of ways.

Printers with resolutions other than 300 dpi are also supported, even if they have different resolutions in the horizontal and vertical directions. High resolution output is supported for typesetters, including an option that compresses the bitmapped fonts so that typesetter virtual memory is not exhausted. This option also significantly reduces the size of the PostScript file and decoding in the printer is very fast.

Missing fonts can be automatically generated if METAFONT exists on the system, or fonts can be converted from '`gf`' to '`pk`' format on demand. If a font cannot be generated, a scaled version of the same font at a different size can be used instead, although dvips will complain loudly about the poor aesthetics of the resulting output.

Users will appreciate features such as collated copies and support for '`tpic`', '`psfig`', '`emtex`', and '`METAPOST`'; system administrators will love the support for multiple printers, each with their own configuration file, and the ability to pipe the output directly to a program such as '`lpr`'. Support for MS-DOS, OS/2, and VMS in addition to UNIX is provided in the standard distribution, and porting to other systems is easy.

One of the most important features is the support of virtual fonts, which add an entirely new level of flexibility to TeX. Virtual fonts are used to give dvips its excellent PostScript font support, handling all the font remapping in a natural, portable, elegant, and extensible way. Dvips even comes with its own Afm2tfm program that creates the necessary virtual fonts and TeX font metric files automatically from the Adobe font metric files.

Source is provided and freely distributable, so adding a site-specific feature is possible. Adding such features is made easier by the highly modular structure of the program.

There is really no reason to use another driver, and the more people use dvips, the less time will be spent fighting with PostScript and the more time will be available to create beautiful documents. So if you don't use dvips on your system, get it today.

2 Introduction to Dvips

To use dvips, simply type

```
dvips foo
```

where '`foo.dvi`' is the output of TeX that you want to print. If dvips has been installed correctly, the document should come out of your default printer.

If you use fonts that have not been used on your system before, they may be automatically generated; this process can take a few minutes. The next time that document is printed, these fonts will already exist, so printing will go much faster.

Many options are available (see Section 7.1 [Dvips options], page 37). For a brief summary of available options, just type

```
dvips
```

3 Paper size and landscape orientation

Most TeX documents at a particular site are designed to use the standard paper size (letter size in the United States, A4 in Europe.) The dvips program defaults to these paper sizes and can be customized for the defaults at each site or on each printer.

But many documents are designed for other paper sizes. For instance, you may want to design a document that has the long edge of the paper horizontal. This can be useful when typesetting booklets, brochures, complex tables, or many other documents. This type of paper orientation is called landscape orientation (the 'normal' orientation is portrait). Alternatively, a document might be designed for ledger or A3 paper.

Since the intended paper size is a document design decision, and not a decision that is made at printing time, such information should be given in the TeX file and not on the dvips command line. For this reason, dvips supports a '`papersize`' special. It is hoped that this special will become standard over time for TeX previewers and other printer drivers.

The format of the '`papersize`' special is

```
\special{papersize=8.5in,11in}
```

where the dimensions given above are for a standard letter sheet. The first dimension given is the horizontal size of the page, and the second is the vertical size. The dimensions supported are the same as for TeX; namely, in (inches), cm (centimeters), mm (millimeters), pt (points), sp (scaled points), bp (big points, the same as the default PostScript unit), pc (picas), dd (didot points), and cc (ciceros).

For a landscape document, the '`papersize`' comment would be given as

```
\special{papersize=11in,8.5in}
```

An alternate specification of '`landscape`' is to have a special of the form

```
\special{landscape}
```

This is supported for backward compatibility, but it is hoped that eventually the '`papersize`' comment will dominate.

Of course, using such a command only informs dvips of the desired paper size; you must still adjust the '`hsize`' and '`vsize`' in your TeX document to actually use the full page.

The '`papersize`' special must occur somewhere on the first page of the document.

4 Including PostScript Graphics

Scaling and including PostScript graphics is a breeze—if the PostScript file is correctly formed. Even if it is not, however, the file can usually be accommodated with just a little more work. The most important feature of a good PostScript file—from the standpoint of including it in another document—is an accurate bounding box comment.

4.1 The Bounding Box Comment

Every well-formed PostScript file has a comment describing where on the page the graphic is located, and how big that graphic is. This information is given in terms of the lower left and upper right corners of a box just enclosing the graphic, and is thus referred to as a bounding box. These coordinates are given in PostScript units (there are precisely 72 PostScript units to the inch) with respect to the lower left corner of the sheet of paper.

To see if a PostScript file has a bounding box comment, just look at the first few lines of the file. (PostScript is standard ASCII, so you can use any text editor to do this.) If within the first few dozen lines there is a line of the form

```
%%BoundingBox: 0 1 2 3
```

(with any numbers), chances are very good that the file is Encapsulated PostScript and will work easily with dvips. If the file contains instead a line like

```
%%BoundingBox: (atend)
```

the file is still probably Encapsulated PostScript, but the bounding box (that dvips needs to position the graphic) is at the end of the file and should be moved to the position of the line above. This can be done with that same text editor, or with a simple Perl script.

If the document lacks a bounding box altogether, one can easily be added. Simply print the file. Now, take a ruler, and make the following measurements. All measurements should be in PostScript units, so measure it in inches and multiply by 72. Alternatively, the '`bbfig`' program distributed with dvips in the '`contrib`' directory can be used to automate this process.

From the left edge of the paper to the leftmost mark on the paper is *llx*, the first number. From the bottom edge of the paper to the bottommost mark on the paper is *lly*, the second number. From the left edge of the paper to the rightmost mark on the paper is *urx*, the third number. The fourth and final number, *ury*, is the distance from the bottom of the page to the uppermost mark on the paper.

Now, add a comment of the following form as the second line of the document. (The first line should already be a line starting with the two characters '`%!`'; if it is not, the file probably isn't PostScript.)

`%%BoundingBox:` *llx lly urx ury*

Or, if you don't want to modify the file, you can simply write these numbers down in a convenient place and use them when you import the graphic.

If the document does not have such a bounding box, or if the bounding box is given at the end of the document, please complain to the authors of the software package that generated the file; without such a line, including PostScript graphics can be tedious.

4.2 Using the EPSF Macros

Now you are ready to include the graphic into a TeX file. Simply add to the top of your TeX file a line like

```
\input epsf
```

(or, if your document is in LaTeX or SliTeX, add the '`epsf`' style option, as was done to the following line).

```
\documentstyle[12pt,epsf]{article}
```

This only needs to be done once, no matter how many figures you plan to include. Now, at the point you want to include the file, enter a line such as

```
\epsffile{foo.ps}
```

If you are using LaTeX or SliTeX, you may need to add a '`\leavevmode`' command immediately before the '`\epsffile`' command to get certain environments to work correctly. If your file did not (or does not currently) have a bounding box comment, you should supply those numbers you wrote down as in the following example:

```
\epsffile[100 100 500 500]{foo.ps}
```

(in the same order they would have been in a normal bounding box comment). Now, save your changes and run TeX and dvips; the output should have your graphic positioned at precisely the point you indicated, with the proper amount of space reserved.

The effect of the '`epsffile`' macro is to typeset the figure as a TeX '`vbox`' at the point of the page that the command is executed. By default, the graphic will have its 'natural' width (namely, the width of its bounding box). The TeX box will have depth zero and a 'natural' height. The graphic will be scaled by any '`dvi`' magnification in effect at the time.

Any PostScript graphics included by any method in this document (except '`bop-hook`' and its ilk) are scaled by the current '`dvi`' magnification. For graphics included with '`epsffile`' where the size is given in TeX dimensions, this scaling will produce the correct, or expected, results. For compatibility with old PostScript drivers, it is possible to turn this scaling off with the following TeX command:

```
\special{! /magscale false def}
```

Use of this command is not recommended because it will make the '`epsffile`' graphics the wrong size if global magnification is used in a '`dvi`' document, and it will cause any PostScript graphics to appear improperly scaled and out of position if a '`dvi`' to '`dvi`' program is used to scale or otherwise modify the document.

You can enlarge or reduce the figure by putting

`\epsfxsize=`*dimen*

right before the call to '`epsffile`'. Then the width of the TeX box will be *dimen* and its' height will be scaled proportionately. Alternatively you can force the vertical size to a particular size with

`\epsfysize=`*dimen*

in which case the height will be set and the width will be scaled proportionally. If you set both, the aspect ratio of the included graphic will be distorted but both size specifications will be honored.

By default, clipping is disabled for included EPSF images. This is because clipping to the bounding box dimensions often cuts off a small portion of the figure, due to slightly inaccurate bounding box arguments. The problem might be subtle; lines around the boundary of the image might be half their intended width, or the tops or bottoms of some text annotations might be sliced off. If you want to turn clipping on, just use the command

```
\epsfclipon
```

and to turn clipping back off, use

```
\epsfclipoff
```

A more general facility for sizing is available by defining the '`epsfsize`' macro. You can redefine this macro to do almost anything. This TeX macro is passed two parameters by '`epsffile`'. The first parameter is the natural horizontal size of the PostScript graphic, and the second parameter is the natural vertical size. This macro is responsible for returning the desired horizontal size of the graph (the same as assigning '`epsfxsize`' above).

In the definitions given below, only the body is given; it should be inserted in

`\def\epsfsize#1#2{`*body*`}`

Some common definitions are:

'`epsfxsize`'
: This definition (the default) enables the default features listed above, by setting '`epsfxsize`' to the same value it had before the macro was called.

'`0pt`'
: This definition forces natural sizes for all graphics by setting the width to zero, which turns on horizontal scaling.

'`#1`'
: This forces natural sizes too, by returning the first parameter only (the natural width) and setting the width to it.

'`hsize`'
: This forces all graphics to be scaled so they are as wide as the current horizontal size. (In LaTeX, use '`textwidth`' instead of '`hsize`'.)

'`0.5#1`'
: This scales all figures to half of their natural size.

'`\ifnum#1>\hsize\hsize\else#1\fi`'
: This keeps graphics at their natural size, unless the width would be wider than the current '`hsize`', in which case the graphic is scaled down to '`hsize`'.

If you want TeX to report the size of the figure as a message on your terminal when it processes each figure, give the command

```
\epsfverbosetrue
```

4.3 Header files

Often in order to get a particular graphic file to work, a certain header file might need to be sent first. Sometimes this is even desirable, since the size of the header macros can dominate the size of certain PostScript graphics files. The dvips program provides support for this with the '`header=`' special command. For instance, to ensure that '`foo.ps`' gets downloaded as part of the header material, the following command should be added to the TeX file:

```
\special{header=foo.ps}
```

The dictionary stack will be at the '`userdict`' level when these header files are included.

For these and all other header files (including the headers required by dvips itself and any downloaded fonts), the printer VM budget is debited by some value. If the header file has, in its first 1024 bytes, a line of the form

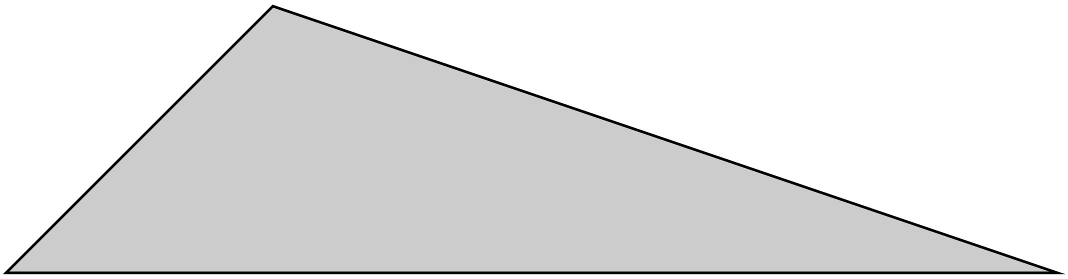
`%%VMusage:` *min max*

then the maximum value is used. If it doesn't, then the total size of the header file in bytes is used as an approximation of the memory requirements.

4.4 Literal PostScript

For simple graphics, or just for experimentation, literal PostScript graphics can be included. Simply use a '`\special`' beginning with a double quote ('`"`') (no closing '`"`').

For instance, the following (simple) graphic:



was created by typing:

```
\vbox to 100bp{\vss % a bp is the same as a PostScript unit
\special{" newpath 0 0 moveto 100 100 lineto 394 0 lineto
closepath gsave 0.8 setgray fill grestore stroke}}
```

(You are responsible for leaving space for such literal graphics.) Literal graphics are discouraged because of their nonportability.

4.5 Literal Headers

Similarly, you can define your own macros for use in such literal graphics through the use of literal macros. Literal macros are defined just like literal graphics, only you begin the special with an exclamation point instead of a double quote. These literal macros are included as part of the header material in a special dictionary called '`SDict`'. This dictionary is the first one on the PostScript dictionary stack when any PostScript graphic is included, whether by literal inclusion or through the '`epsffile`' macros.

4.6 Other Graphics Support

There are other ways to include graphics with dvips. One is to use an existing package, such as '`emtex`', '`psfig`', '`tpic`', or '`METAPOST`', all of which Dvips supports.

Other facilities are available for historical reasons, but their use is discouraged, in hope that some 'sane' form of PostScript inclusion shall become standard. The main advantage of the '`epsffile`' macros is that they can be adapted for whatever form of special eventually becomes standard, and thus only minor modifications to that one file need to be made, rather than revising an entire library of TEX documents.

Most of these specials use a flexible key and value scheme:

```
\special{psfile=filename.ps[key=value]*}
```

This will download the PostScript file called '`filename.ps`' such that the current point will be the origin of the PostScript coordinate system. The optional key/value assignments allow you to specify transformations on the PostScript.

The possible keys are:

'`hoffset`' The horizontal offset (default 0)

'`voffset`' The vertical offset (default 0)

'`hsize`' The horizontal clipping size (default 612)

'`vsize`' The vertical clipping size (default 792)

'`hscale`' The horizontal scaling factor (default 100)

'`vscale`' The vertical scaling factor (default 100)

'`angle`' The rotation (default 0)

'`clip`' Enable clipping to the bounding box

The dimension parameters are all given in PostScript units. The '`hscale`' and '`vscale`' are given in non-dimensioned percentage units, and the rotation value is specified in degrees. Thus

```
\special{psfile=foo.ps hoffset=72 hscale=90 vscale=90}
```

will shift the graphics produced by file '`foo.ps`' right by one inch and will draw it at 0.9 times normal size. Offsets are given relative to the point of the special command, and are unaffected by scaling or rotation. Rotation is counterclockwise about the origin. The order of operations is to rotate the figure, scale it, then offset it.

For compatibility with older PostScript drivers, it is possible to change the units that '`hscale`' and '`vscale`' are given in. This can be done by redefining '`@scaleunit`' in '`SDict`' by a TeX command such as

```
\special{! /@scaleunit 1 def}
```

The '`@scaleunit`' variable, which is by default 100, is what '`hscale`' and '`vscale`' are divided by to yield an absolute scale factor.

All of the methods for including graphics we have described so far enclose the graphic in a PostScript save/restore pair, guaranteeing that the figure will have no effect on the rest of the document. Another type of special command allows literal PostScript instructions to be inserted without enclosing them in this protective shield; users of this feature are supposed to understand what they are doing (and they shouldn't change the PostScript graphics state unless they are willing to take the consequences). This command can take many forms, because it has had a torturous history; any of the following will work:

```
\special{ps:text}
\special{ps::text}
\special{ps::[begin]text}
\special{ps::[end]text}
```

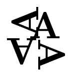
(with longer forms taking precedence over shorter forms, when they are used). Note that '`ps::`' and '`ps::[end]`' do not do any positioning, so they can be used to continue PostScript literals started with '`ps:`' or '`ps::[begin]`'. There is also the command

```
\special{ps: plotfile filename}
```

which will copy the commands from *filename* verbatim into the output (but omitting lines that begin with %). An example of the proper use of literal specials can be found in the file '`rotate.tex`' which makes it easy to typeset text turned 90 degrees.

To finish off this section, the following examples are presented without explanation:

```
\def\rotninety{\special{ps:currentpoint currentpoint translate 90
rotate neg exch neg exch translate}}\font\huge=cmbx10 at 14.4truept
\setbox0=\hbox to0pt{\huge A\hss}\vskip16truept\centerline{\copy0
\special{ps:gsave}\rotninety\copy0\rotninety\copy0\rotninety
\box0\special{ps:grestore}}\vskip16truept
```



```
\vbox to 2truein{\special{ps:gsave 0.3 setgray}\hrule height 2in
width\hsize\vskip-2in\special{ps:grestore}\font\big=cminch\big
\vss\special{ps:gsave 1 setgray}\vbox to 0pt{\vskip2pt
\line{\hss\hskip4pt NEAT\hss}\vss}\special{ps:0 setgray}%
\hbox{\raise2pt\line{\hss NEAT\hss}\special{ps:grestore}}\vss}
```

Some caveats are in order when using the above forms. Make sure that each '`gsave`' on a page is matched with a '`grestore`' on the same page. Do not use '`save`' or '`restore`'. Use of these macros can interact with the PostScript generated by dvips if care is not taken; try to understand what the above macros are doing before writing your own. The '`\rotninety`' macro especially has a useful trick that appears again and again.

4.7 Dynamic Creation of PostScript Graphics Files

PostScript is an excellent page description language—but it does tend to be rather verbose. Compressing PostScript graphics files can often reduce them by more than a factor of five. For this reason, if the filename parameter to one of the graphics inclusion techniques starts with a backtick ('`` ` ``'), the filename is instead interpreted as a command to execute that will send the actual file to standard output. Thus,

```
\special{psfile="`zcat foo.ps.Z"}
```

will include the uncompressed version of '`foo.ps`'. Since such a command is not accessible to TeX, if you use this facility with the '`EPSF`' macros, you need to supply the bounding box parameter yourself, as in

```
\epsffile[72 72 540 720]{"`zcat screendump.ps.Z"}
```

to include '`screendump.ps`'. Of course, the commands to be executed can be anything, including using a file conversion utility such as '`tek2ps`' or whatever is appropriate.

This extension is not portable to other DVI-to-PostScript translators.

5 PostScript fonts

Dvips supports the use of PostScript fonts in TeX documents. To use a PostScript font conveniently, you need to prepare a corresponding virtual font; the program Afm2tfm, supplied with Dvips, helps with that.

All the necessary support for the standard 35 PostScript fonts (`AvantGarde-Book` through `ZapfDingbats`), plus other freely or commonly available PostScript fonts, comes with Dvips. To use those fonts, you need do nothing beyond the normal Dvips installation (see Chapter 9 [Installation], page 50). This chapter is therefore relevant only if you are installing new PostScript fonts not supplied with Dvips. (Or if you're curious.)

(In fact, some the fonts supplied with Dvips contain a few modifications to make them work as well as possible with TeX. So don't expect that following the steps in ths manual will necessarily generate identical files to those in the distribution.)

5.1 Font concepts

All the information about an ordinary font that's needed to typeset the font is contained in two files: a *metric file* that contains shape-independent information and a *glyph file* that contains the actual shapes of the font's characters.

A *virtual font* is an additional file that specifies where the characters in the font come from, or how they are constructed. TeX itself (and LaTeX) only look at the metric file, but DVI drivers such as Dvips look at all three of these files.

An *encoding file* records the correspondence between the code numbers of the characters in a font and their descriptive names. Two encoding files used together can describe a reencoding that rearranges, i.e., renumbers, the characters of a font.

5.1.1 Metric files

A *metric file* describes properties of the font that are independent of what the characters actually look like. Aside from general information about the font itself, a metric file has two kinds of information: information about individual characters, organized by character code, and information about sequences of characters.

The per-character information specifies the width and height of each character in the font. It also specifies each character's depth and italic correction if those are nonzero.

In addition to information on individual characters, the metric file specifies *kerning*, i.e., adding or removing space between particular character pairs. It further specifies *ligature* information: when a sequence of input characters should be typeset as a different character, namely, the ligature character. For example, it's traditional for the input '`fi`' to be typeset as 'fi', not as 'fi'. (In English, the only common ligatures are fi, fl, ff, ffi, and ffl.)

Different typesetting systems use different formats for their metrics files:

- Each Postscript font has an *Adobe font metrics* ('`.afm`') file. These files are plain text, so you can inspect them easily. The AFM files for most of Adobe's fonts are publicly available and redistributable, but the files defining the actual shapes are not publicly available (you have to buy those from Adobe). You can get Adobe's AFM files by ftp from '`ftp.mv.us.adobe.com`'.
- TeX uses *TeX font metrics* ('`.tfm`') files. When you say '`\font =` *font*' in your TeX document, TeX looks for a file named '*font*`.tfm`'. TeX uses that file to calculate the space occupied by characters from the font. In addition, the DVI drivers you use to print or view the DVI file produced by TeX may need to look at the TFM file.

  TFM files are binary (and hence are typically much smaller than AFM files). You can use the `tftopl` program that comes with TeX to transform a TFM file into a human-readable "property list" ('`.pl`') file. You can also edit a PL file and transform it back to a TeX-readable TFM with the companion program `pltotf`. (Editing metrics by hand is not something you're likely to want to do often, but the capability is there.)
- ATM and other typesetting systems use *printer font metric* ('`.pfm`') files. These are binary files. They are irrelevant in the TeX world, and not freely available, so we will not discuss them further.

The Afm2tfm program distributed with Dvips converts an AFM file to a TFM file and performs other useful transformations as well.

5.1.2 Glyph files

Although a metric file (see the previous section) contains information about the spatial and other properties of the character at position 75, say, it contains nothing about what the character at position 75 actually looks like. The *glyphs*—the actual shapes of the letterforms in a font—are defined by other files. TeX itself reads only the TFM file for a font; it does not need to know what the glyphs of the characters are.

A *glyph file* is a file that defines the shapes of the characters in a font. The shapes can be defined either by outlines or by bitmaps.

PostScript fonts are defined as *outline fonts*: Each character in the font is defined by giving the mathematical curves (lines, arcs, and splines) that define its contours. Different sizes of a character are generated by linearly scaling a single shape. For example, a 10-point 'A' is simply half the size of a 20-point 'A'. Nowadays, outline fonts usually also contain *hints*—additional information that modifies the outline slightly to improve the appearance of the font at small sizes or low resolutions.

Although there are various kinds of PostScript outline fonts, by far the most common is *Type 1* fonts. For our purposes, the differences between the types are unimportant. The glyph files for Postscript Type 1 fonts typically have names ending in '`.pfa`' ("printer font ascii") or '`.pfb`' ("printer font binary").

In contrast, Computer Modern and the other standard TeX fonts are defined as *bitmapped fonts*, generated from Metafont ('`.mf`') descriptions. The Metafont program distributed with TeX generates bitmapped fonts from these descriptions, which are essentially outlines.

The glyph files for TeX bitmapped fonts are usually stored in *packed font* (PK) files. The names of these files end in '`.pk`*n*', where *n* is the resolution of the font in dots per inch. For example, '`cmr10.300pk`' contains the bitmap for the '`cmr10`' font at 300 dpi resolution.

Metafont actually outputs *generic font* (GF) files, e.g., '`cmr10.300gf`', but the GF files are usually converted immediately to PK format (using the `gftopk` utility that comes with TeX) since PK files are smaller and contain the same information. (The GF format is a historical artifact.)

5.1.3 Virtual fonts

A *virtual font* is constructed by extracting characters from one or more existing fonts and rearranging them, or synthesizing new characters in various ways. The explanation in this manual is intended to suffice for understanding enough about virtual fonts to use them with Dvips. It isn't a reference manual on virtual fonts.

A virtual font ('`.vf`') file specifies, for each character in the virtual font, a recipe for typesetting that character. A VF file, like a TFM file, is in a compressed binary format. The `vftovp` and `vptovf` programs convert a VF file to a human-readable VPL (virtual property list) format and back again.

In the case of a PostScript font f being used in a straightforward way, the recipe says: "character i in the VF font is character j in font f". The font f is called a *base font*. Here, the only function of the VF file is to remap the characters of the PostScript font to the positions where TEX expects to find them. See Section 5.1.4 [Encodings], page 18.

Since TEX reads only TFM files, not VF's, each VF must have a corresponding TFM for use with TEX. This corresponding TFM is created when you run `vptovf`.

For more information: The primary document on virtual fonts is Donald E. Knuth, *TUGboat* 11(1), Apr. 1990, pp. 13–23, "Virtual Fonts: More Fun for Grand Wizards". (Unfortunately, the subtitle of that article alone would intimidate almost anyone.)

5.1.4 Encodings

Every font, whatever its type, has an *encoding*, that specifies the correspondence between "logical" characters and character codes. For example, the ASCII encoding specifies that the character numbered 65 (decimal) is an uppercase 'A'. The encoding does *not* specify what the character at that position looks like; there are lots of ways to draw an 'A', and a glyph file (see Section 5.1.2 [Glyph files], page 16) tells how. Nor does it specify how much space that character occupies; that information is in a metric file (see Section 5.1.1 [Metric files], page 15).

TEX implicitly assumes a particular encoding for the fonts you use with it. For example, the plain TEX macro `\'`, which typesets an acute accent over the following letter, assumes the acute accent is at position 19 (decimal). This happens to be true of the standard TEX fonts, as you might expect, but it is *not* true of normal PostScript fonts.

It's possible but painful to change all the macros that assume particular character positions. A better solution is to create a new font with the information for the acute accent at position 19, where TEX expects it to be. See Section 5.2 [Making a font available], page 20.

One particular way to represent an encoding is as a sequence of 256 character names called an *encoding vector*. An *encoding file* ('`.enc`') stores this vector, together with ligature and kerning information (with which we are not concerned at the moment). These encoding files are used by the Afm2tfm program. Encoding files are also downloaded to the PostScript interpreter in your printer if you use one of them in place of the default encoding vector for a particular PostScript font.

Examples of encodings: the 'dvips.enc' encoding file that comes with Dvips is a good (but not perfect) approximation to the TEX encoding for TEX's Computer Modern text fonts. The distribution also includes three other encoding files corresponding to the TEX mathematics fonts: 'texmext.enc' for math extensions, 'texmital.enc' for math italics, and 'texmsym.enc' for math symbols.

5.1.5 How PostScript typesets a character

The output of Dvips is a program in the PostScript language that instructs your (presumably PostScript-capable) printer how to typeset your document by transforming it into toner on paper. Your printer, in turn, contains a PostScript interpreter that carries out the instructions in this typesetting program.

Part of the program is the definition of any PostScript fonts that you use in your document. Fonts that come with your printer (probably the "standard 35", including 'Times-Roman', 'Symbol', ...) are always defined. Other fonts must be downloaded as pfa or pfb files (see Section 5.1.2 [Glyph files], page 16) from your computer. (The one on which you're running Dvips.)

You may be wondering exactly how a PostScript interpreter figures out what character to typeset, with this mass of metrics, glyphs, encodings, and other information. If you're not wondering, skip this section ...

The basic PostScript operator for imaging characters is `show`. Suppose you've asked TEX to typeset an 'S'. This will eventually wind up in the Dvips output as something equivalent to this:

```
(S) show
```

Here is how PostScript typesets the S:

1. PostScript interpreters use ASCII; therefore 'S' is represented as the integer 83. (Any of the 256 possible characters representable in a standard 8-bit byte can be typeset.)
2. A PostScript *dictionary* is a mapping of names to arbitrary values. A font, to the interpreter, is a dictionary which contains entries for certain names. (If these entries are missing, the interpreter refuses to to do anything with that font.)

   PostScript has a notion of "the current font"—whatever font is currently being typeset in.
3. One of these mandatory entries in a font dictionary is 'Encoding', which defines the encoding vector (see Section 5.1.4 [Encodings], page 18) for that font. This vector of 256 names maps each possible input character to a name.

4. The interpreter retrieves the entry at position 83 of the encoding vector. This value is a PostScript name: `/S`.
5. For Type 1 fonts (we're not going to discuss anything else), the interpreter now looks up `/S` as a key in a dictionary named `CharStrings`. ('`CharStrings`' is another of the mandatory entries in a font dictionary.)
6. The value of `S` in `CharStrings` is the equivalent of a series of standard PostScript commands like '`curveto`', '`lineto`', '`fill`', and so on. These commands are executed to draw the character. There can also be *hint* information that helps adapt the character to low-resolution rasters. (See Section 5.1.2 [Glyph files], page 16.)

   (The commands are actually represented in a more compact way than standard PostScript code; see the Type 1 book for details.)

This method for typesetting characters is used in both Level 1 and Level 2 PostScript. See the PostScript reference manuals for more information.

5.2 Making a PostScript font available

To make a PostScript font available in a TeX document, you need to install the font on your system and then define it within the document. Once you have installed the font, of course, it is available for any document thereafter and you don't need to reinstall it. You must have an AFM file for any font you install. Unless the font is built into your printer, you must also have a PFA or PFB file.

In the following examples, we use the font '`Times-Roman`' to illustrate the process. (But you should use the files for Times supplied with Dvips for actual use, as explained above.)

Installation of a PostScript font proceeds in three steps. See Section 5.1 [Font concepts], page 15, for descriptions of the various files involved.

1. Run `afm2tfm` to create a TFM file for the original font, and a VPL file for the corresponding virtual font. See Section 5.3 [Invoking afm2tfm], page 22.
2. Run `vptovf` to generate a VF and TFM file for the virtual font from the VPL file.
3. Insert entries for the font in the '`psfonts.map`' file. See Section 5.4 [psfonts.map], page 29.

The typical invocation of Afm2tfm goes something like this:

```
afm2tfm Times-Roman -v ptmr rptmr
```

This produces two files as output, namely the 'virtual property list' file '`ptmr.vpl`', and the raw font metric file '`rptmr.tfm`'. (The uppercase '`-V`' also produces a '`vpl`' file, but maps a set of small caps into the lower case alphabet.) To use the font in TeX, you first run

```
vptovf ptmr.vpl ptmr.vf ptmr.tfm
```

You should then install the virtual font file '`ptmr.vf`' where Dvips will see it and install '`ptmr.tfm`' and '`rptmr.tfm`' where TeX and Dvips will see them.

You can also make more complex virtual fonts by editing '`ptmr.vpl`' before running '`vptovf`'; such editing might add the uppercase Greek characters in the standard TeX positions, for instance. (This has already been done for the font files supplied with Dvips.)

Once this has been done, you're all set. You can use code like this in TeX henceforth:

```
\font\myfont = ptmr at 12pt
\myfont Hello, I am being typeset in 12-point Times-Roman.
```

Thus we have two fonts, one actual ('`rptmr`', which is analogous to the font in the printer) and one virtual ('`ptmr`', which has been remapped to the standard TeX encoding (almost), and has typesetting know-how added. You could also say

```
\font\PTR = rptmr at 10pt
```

and typeset directly with that, but then you would have no ligatures or kerning, and you would have to use Adobe character positions for special letters like Æ. The virtual font '`ptmr`' not only has ligatures and kerning, and most of the standard accent conventions of TeX, it also has a few additional features not present in the Computer Modern fonts. For example, it includes all the Adobe characters (such as the Polish ogonek and the French guillemots). The only things you lose from ordinary TeX text fonts are the dotless 'j' (which can be hacked into the VPL file with literal PostScript specials if you have the patience) and uppercase Greek letters (which just don't exist unless you buy them separately).

As a final step you need to record information about both the virtual font and the original font (if you ever might want to use it) in the '`psfonts.map`' file. See Section 5.4 [psfonts.map], page 29. For our example, you'd insert the following into '`psfonts.map`':

```
rptmr      Times-Roman                  <rptmr.pfa
```

These PostScript fonts can be scaled to any size. Go wild! Using PostScript fonts, however, does use up a great deal of the printer's memory and it does take time. You may find downloading bitmapped fonts (possibly compressed) to be faster than using the built-in PostScript fonts.

5.3 Invoking Afm2tfm

The Afm2tfm program converts an AFM file for a PostScript font to a TFM file and a VPL file for a corresponding virtual font (or, in its simplest form, to a TFM file for the PostScript font itself). The results of the conversion are affected by the command-line options and especially by the reencodings you can specify with those options. You can also obtain special effects such as an oblique font.

5.3.1 Changing font encodings

Afm2tfm allows you to specify a different encoding for a PostScript font (for a general introduction to encodings, see Section 5.1.4 [Encodings], page 18). You do this with '`-t`' for the TeX encoding, '`-p`' for the PostScript encoding, or with '`-T`' for both simultaneously.

5.3.1.1 Changing TeX encodings

If you are using Afm2tfm to build a virtual font, i.e., you have specified the '`-v`' or '`-V`' option, you can then specify an encoding for that virtual font with '`-t` *tex-enc*'. ('`-t`' is ignored if neither '`-v`' nor '`-V`' is present.) Any ligature and kerning information you specify in *tex-enc* will be used in the VPL, in addition to the ligature and kerning information from the AFM file.

If the AFM file has no entry for a character specified in *tex-enc*, that character will be omitted from the VPL.

The '`-t`' option is likely to be needed when you have a PostScript font corresponding to a TeX font other than a normal text font such as Computer Modern. For instance, if you have a PostScript font that contains math symbols, you'd probably want to use the encoding in the '`texmsym.enc`' file supplied with Dvips. (For a start; to actually get usable math fonts, you have to define much more than just an encoding.)

5.3.1.2 Changing PostScript encodings

By default, Afm2tfm uses the encoding it finds in the AFM file. You can specify a different PostScript encoding with '`-p` *ps-enc*'. This makes the raw TFM file (the one output by Afm2tfm) have the encoding specified in the encoding file *ps-enc*. Any ligature or kern information specified in *ps-enc* is ignored by Afm2tfm, since ligkern info is always omitted from the raw TFM anyway.

If you use this option, you must also arrange to download *ps-enc* as part of any document that uses this font. You do this by adding a line like the following one to '`psfonts.map`' (see Section 5.4 [psfonts.map], page 29):

```
zpopr Optima "MyEncoding ReEncodeFont" <myenc.enc
```

Using '`-p`' is the only way to access characters in a PostScript font that are neither encoded in the AFM file nor constructed from other characters. For instance, Adobe's '`Times-Roman`' font contains the extra characters '`trademark`' and '`registered`' (among others); these can only be accessed through such a PostScript reencoding.

5.3.1.3 Changing both TEX and PostScript encodings

The option '`-T` *enc-file*' is equivalent to '`-p` *enc-file* `-t` *enc-file*'. If you make regular use of a private non-standard reencoding '`-T`' is usually the better idea, since you can get unexpected inconsistencies in mapping otherwise. An example of when you might use this option is when you have a TEX encoding that is designed to be used with a particular PostScript font, e.g., a dingbats font.

5.3.1.4 Reencoding with Afm2tfm

The Afm2tfm program creates the TFM and VF files for the virtual font corresponding to a PostScript font by *reencoding* the PostScript font. Afm2tfm generates these files from two encodings: one for TEX and one for PostScript. The TEX encoding is used to map character numbers to character names while the PostScript encoding is used to map each character name to a possibly different number. In combination, you can get access to any character of a PostScript font at any position for TEX typesetting.

In the default case, when you specify none of the `-t`, `-p`, or `-T` options, Afm2tfm uses a default TEX encoding (which mostly corresponds to the Computer Modern text fonts) and the default

PostScript encoding (the one found in the AFM file that came with the font). The reencoding is also sometimes called a *remapping*.

For example, the default encodings reencode the acute accent in two steps: first the default TEX encoding maps the number 19 to the character name '`acute`'; then the default PostScript encoding, as found in the AFM file for an ordinary PostScript font, maps the character name '`acute`' to the number 194. (The PostScript encoding works in reverse, by looking in the encoding vector for the name and then yielding the corresponding number.) The combined mapping of 19 to 194 shows up explicitly in the VF file and also implicitly in the fact that the properties of PostScript character 194 appear in position 19 of the TFM file for the virtual font.

The remapped `p*` fonts mostly follow plain TEX conventions for accents. The exceptions: the Hungarian umlaut (which is at position '`0x7D`' in '`cmr10`', but position '`0xCD`' in '`ptmr`'); the dot accent (at positions '`0x5F`' and '`0xC7`', respectively); and '`\AA`', which neds different tweaking. In order to use these accents with PostScript fonts or in math mode when '`\textfont0`' is a PostScript font, you will need to use the following definitions. These definitions will not work with the Computer Modern fonts for the relevant accents. They are already part of the distributed '`psfonts.sty`' for use with LaTEX.

```
\def\H#1{{\accent"CD #1}}
\def\.#1{{\accent"C7 #1}}
\def\dot{\mathaccent"70C7 }
\newdimen\aadimen
\def\AA{\leavevmode\setbox0\hbox{h}\aadimen\ht0
  \advance\aadimen-1ex\setbox0\hbox{A}\rlap{\raise.67\aadimen
  \hbox to \wd0{\hss\char'27\hss}}A}
```

5.3.1.5 Encoding file format

Afm2tfm's encoding files have the same format as an encoding vector in a PostScript font. Here is a skeletal example:

```
% Comments are ignored, unless the first word after the percent sign
% is 'LIGKERN'—-see below.
/MyEncoding [ % exactly 256 entries follow, each with a leading '/'
  /Alpha /Beta /Gamma /Delta ...
  /A /B ... /Z
  ...  /.notdef /xfooaccent /yfooaccent /zfooaccent
] def
```

These encoding files are downloaded as part of changing the encoding at the PostScript level (see the previous section).

Comments, which start with a percent sign and continue until the end of the line, are ignored unless they start with 'LIGKERN'.

The first non-comment word of the file must start with a forward slash '/' (i.e., a PostScript literal name) and defines the name of the encoding. The next word must be an left bracket '['. Following that must be precisely 256 character names; use '/.notdef' for any that you want to leave undefined. Then there must be a matching right bracket]. A final 'def' token is optional. All names are case-sensitive.

Any ligature or kern information is given as a comment. If the first word after the '%' is 'LIGKERN', then the entire rest of the line is parsed for ligature and kern information. This ligature and kern information is given in groups of words, each group of which must be terminated by a semicolon (with a space before and after it, unless it occurs at the end of a line).

In these 'LIGKERN' statements, three types of information may be specified. These three types are ligature pairs, kerns to ignore, and the character value of this font's boundary character.

Throughout a 'LIGKERN' statement, the boundary character is specified as '||'. To set the boundary character value:

```
% LIGKERN || = 39 ;
```

To indicate a kern to remove, give the names of the two characters (without the leading slash) separated by '{}', as in 'one {} one ;'. This is similar to the way you might use '{}' in a TeX file to turn off ligatures or kerns at a particular location. Either or both of the character names can be given as '*', which is a wild card matching any character; thus, all kerns can be removed with '* {} * ;'.

To specify a ligature, specify the names of the pair of characters, followed by the ligature 'operation' (as in Metafont), followed by the replacing character name. Either (but not both) of the first two characters can be '||' to indicate a word boundary.

Typically the operation is '=:' meaning that both characters are removed and replaced by the third character, but by adding '|' characters on either side of the '=:', you can specify which of the two leading characters to retain. In addition, by suffixing the ligature operation with one or two '>' signs, you can indicate that the ligature scanning operation should skip that many characters before proceeding. This works just like in METAFONT. A typical ligature might be specified with '`ff i =: ffi ;`'. A more convoluted ligature is '`one one |=:|>> exclam ;`' which indicates that every pair of adjacent '`1`''s should be separated by an exclamation point, and then two of the resulting characters should be skipped over before continuing searching for ligatures and kerns. You cannot give more '`>`''s in an ligature operation as you did '`|`', so there are a total of eight possible ligature operations:

```
=: |=: |=:> =:| =:|> |=:| |=:|> |=:|>>
```

The default set of ligatures and kerns built in to Afm2tfm can be specified with:

```
% LIGKERN question quoteleft =: questiondown ;
% LIGKERN exclam quoteleft =: exclamdown ;
% LIGKERN hyphen hyphen =: endash ; endash hyphen =: emdash ;
% LIGKERN quoteleft quoteleft =: quotedblleft ;
% LIGKERN quoteright quoteright =: quotedblright ;
% LIGKERN space {} * ; * {} space ; 0 {} * ; * {} 0 ;
% LIGKERN 1 {} * ; * {} 1 ; 2 {} * ; * {} 2 ; 3 {} * ; * {} 3 ;
% LIGKERN 4 {} * ; * {} 4 ; 5 {} * ; * {} 5 ; 6 {} * ; * {} 6 ;
% LIGKERN 7 {} * ; * {} 7 ; 8 {} * ; * {} 8 ; 9 {} * ; * {} 9 ;
```

5.3.2 Special font effects

Besides the reencoding described in the previous section Afm2tfm can do other manipulations.

For example, to make an obliqued variant of Times Roman:

```
afm2tfm Times-Roman -s .167 -v ptmro rptmro
```

which creates '`ptmro.vpl`' and '`rptmro.tfm`'. To use this, put the line

```
rptmro Times-Roman ".167 SlantFont"
```

into '`psfonts.map`'. Then '`rptmro`' (our name for the obliqued Times) will act as if it were a resident font, although it is actually constructed from Times-Roman via the PostScript routine `SlantFont` (which slants everything 1/6 to the right, in this case).

Similarly, you can get an expanded font with

```
afm2tfm Times-Roman -e 1.2 -v ptmrre rptmrre
```

and by recording the pseudo-resident font

```
rptmrre Times-Roman "1.2 ExtendFont"
```

in 'psfonts.map'.

You can also create a small caps font with a command such as

```
afm2tfm Times-Roman -V ptmrc rptmrc
```

This will generate a set of small caps mapped into the usual lower case positions and scaled down to 0.8 of the normal cap dimensions. You can also specify the scaling as something other than the default 0.8:

```
afm2tfm Times-Roman -c 0.7 -V ptmrc rptmrc
```

It is unfortunately not possible to increase the width of the small caps independently of the rest of the font. If you want a really professional looking set of small caps you will have to make up a custom pair of TFM and VPL files using the '-e' option, change the name and checksum of the D 1 font in 'ptmrc.vpl' to match the customized raw TFM, and replace the small caps at the lower case mappings with the small caps in the customized file. You can then throw away the customized VPL file, but not the customized 'tfm'. That must be identified with an appropriate line in 'psfonts.map'.

To change the PaintType in a font from filled (0) to outlined (2), you can add "/PaintType 2 store" to 'psfonts.map', as in the following:

```
rphvrl   Helvetica "/PaintType 2 store"
```

The Afm2tfm program prints out the precise line you need to add to 'psfonts.map' to use that font, assuming the font is resident in the printer; if the font is not resident, you must add the header command to download the font yourself. Each identical line only needs to be specified once in the 'psfonts.map' file, even though many different fonts (small caps variants, or ones with different output encodings) may be based on it.

5.3.3 afm2tfm options

Synopsis:

afm2tfm [*option*]... *afmfile*[.afm] [*tfmfile*[.tfm]]

Afm2tfm reads *afmfile* and writes a corresponding (but raw) TFM file. If *tfmfile* is not supplied, the base name of the AFM file is extended with '.tfm' to get the output filename.

In the simplest case you could use Afm2tfm to convert an Adobe AFM file such as 'Times-Roman.afm' into a "raw" TEX TFM file such as 'rptmr.tfm' with a command such as

```
afm2tfm Times-Roman rptmr
```

The TFM file thus created is raw because it does no character remapping; it simply contains the character information in the AFM file in TFM form, which is the form that TEX understands. The characters have the same code in the TFM file as in the AFM file. The 96 printable ASCII characters will be described correctly in the TFM file but the other characters in the font will not be.[1] That is because standard PostScript fonts have a different encoding scheme than the one that plain TEX expects. See Section 5.1.4 [Encodings], page 18. Although both schemes agree for the printable ASCII characters, special characters such as ligatures and accents are numbered quite differently.

To enable TEX to use those special characters, you create a virtual font as well, with the '-v' or '-V' option. See Section 5.2 [Making a font available], page 20.

The command line switches to Afm2tfm are:

'-c *ratio*' See '-V'; overrides the default ratio of 0.8 for the scaling of small caps.

'-e *ratio*' All characters are stretched horizontally by the stated ratio; if it is less than 1.0, you get a condensed font.

'-O' This option forces all character codes in the output 'vpl' file be given as octal values; this is useful for symbol or other special-purpose fonts where character names such as 'A' have no meaning.

[1] It would be better to have eschewed the use of the 'r' prefix in favor of using a variant letter to specify the encoding; but we didn't think of that at the time, and it's too late to change existing fonts. See section "Top" in *Filenames for TEX fonts*.

'-p *file*' This specifies a file to use for the PostScript encoding of the font. This file must also be mentioned as a header file for the font in '`psfonts.map`'. See Section 5.3.1.2 [Changing PostScript encodings], page 23.

'-s *slant*' All characters are slanted to the right by *slant*. If *slant* is negative, the letters slope to the left (or they might be upright if you start with an italic font).

'-t *file*' This specifies a file to use for the target TEX encoding of the font. Ligature and kern information may also be specified in *file*. *file* is not mentioned in '`psfonts.map`'.

'-T *file*' This option specifies that *file* is to be used for both the PostScript and target TEX encodings of the font. Equivalent to '`-p` *file* `-t` *file*'.

'-u' Have Afm2tfm use only those characters that are required by the TEX encoding, and no others. Normally, Afm2tfm tries to include both characters that fit the TEX encoding and any additional characters that might exist in the font. This option stops those additional characters from being added.

'-v *file*' Generate a VPL (virtual property list) file as well as a TFM file.

'-V *file*' Same as '`-v`', but the virtual font generated is a small caps font obtained by scaling uppercase letters by 0.8 to typeset lowercase. This font handles accented letters and retains proper kerning.

Here are the '`CODINGSCHEME`'s that result from the various possible choices for reencoding.

'`default encoding`'
`(CODINGSCHEME TeX text + AdobeStandardEncoding)`

'`-p EC.enc`'
`(CODINGSCHEME TeX text + ECEncoding)`

'`-t EC.enc`'
`(CODINGSCHEME ECEncoding + AdobeStandardEncoding)`

'`-T EC.enc`'
`(CODINGSCHEME ECEncoding + ECEncoding)`

The '`CODINGSCHEME`' line appears in the VPL file but is ignored by Dvips.

5.4 The PostScript font map file '`psfonts.map`'

The '`psfonts.map`' file associates a font with its files. For example, the line

```
rpstrn StoneInformal <StoneInformal.pfb
```

causes Dvips to include 'StoneInformal.pfb' in your document as if it were a header file whenever the pseudo-resident font 'StoneInformal' is used in a document Section 4.3 [Header files], page 10. Similarly, you can generate transformed fonts (see Section 5.3.2 [Special font effects], page 26) and include a line like

```
rpstrc StoneInformal <StoneInformal.pfb ".8 ExtendFont"
```

in 'psfonts.map'. In this case 'StoneInformal.pfb' will be loaded whenever you use the StoneInformal-Condensed font. Each header file is loaded at most once per document.

If you are using a 'pfb' file that has a nonstandard PostScript encoding, you need to include a reference to the corresponding encoding file in 'psfonts.map'. If, for instance, you want to reencode the PostScript font 'Optima' with the encoding 'MyEncoding', stored in 'myenc.enc', you need:

```
rpstrnx Optima "MyEncoding ReEncodeFont" <myenc.enc <Optima.pfb
```

When using PFB files, Dvips is smart enough to unpack the binary PFB format into printable ASCII so there is no need to perform this conversion yourself. In addition, Dvips scans the font to determine its memory usage, just as it does for any header file.

Here is a brief description of the contents of 'psfonts.map'. If a line is empty or begins with a space, percent, asterisk, semicolon, or hash mark, it is ignored. Otherwise, the line is separated into words, where words are separated by spaces or tabs, except that if a word begins with a double quote, it extends until the next double quote or the end of the line. If a word starts with a less than character, it is treated as a font header file (or a downloaded PostScript font). There can be more than one such header for a given font. If a word starts with a double quote, it is a special instruction for generating that font. Otherwise it is a name. The first such name is the TFM file that a virtual font file can refer to. If there is another name, it is used as the PostScript name; if there is only one name, it is used for both the TeX name and the PostScript name.

6 Using Color with dvips

This new feature of Dvips is somewhat experimental so your experiences and comments are welcome. Initially added by Jim Hafner, IBM Research, '`hafner@almaden.ibm.com`', the color support has gone through many changes by Tomas Rokicki. Besides the changes to the source code itself, there are additional TEX macro files: '`colordvi.tex`' and '`blackdvi.tex`'. There are also '`.sty`' versions of these files that can be used with LaTEX and other similar macro packages. This feature adds one-pass multi-color printing of TEX documents on any color PostScript device.

In this section we describe the use of color from the document preparer's point of view and then add some simple instructions on installation for the system administrator.

6.1 Macro Files

All the color macro commands are defined in '`colordvi.tex`' (or '`colordvi.sty`'). To access these macros simply add to the top of your TEX file the command

```
\input colordvi
```

or, if your document uses style files like LaTEX, add the '`colordvi`' style option as in

```
\documentstyle[12pt,colordvi]{article}
```

There are basically two kinds of color macros, ones for local color changes to, say, a few words or even one symbol and one for global color changes. Note that all the color names use a mixed case scheme. There are 68 predefined colors, with names taken primarily from the Crayola crayon box of 64 colors, and one pair of macros for the user to set his own color pattern. More on this extra feature later. You can browse the file '`colordvi.tex`' for a list of the predefined colors. The comments in this file also show a rough correspondence between the crayon names and PANTONEs.

A local color command is in the form

```
\ColorName{this will print in color}
```

Here '`ColorName`' is the name of a predefined color. As this example shows, this type of command takes one argument which is the text that is to print in the selected color. This can be used for nested color changes since it restores the original color state when it completes. For example, suppose you were writing in green and want to switch temporarily to red, then blue, back to red and restore green. Here is one way that you can do this:

```
This text is green but here we are \Red{switching to red,
\Blue{nesting blue}, recovering the red} and back to
original green.
```

In principle there is no limit to the nesting level, but it is not advisable to nest too deep lest you loose track of the color history.

The global color command has the form

```
\textColorName
```

This macro takes no arguments and immediately changes the default color from that point on to the specified color. This of course can be overriden globally by another such command or locally by local color commands. For example, expanding on the example above, we might have

```
\textGreen
This text is green but here we are \Red{switching to red,
\Blue{nesting blue}, recovering the red} and back to
original green.
\textCyan
The text from here on will be cyan unless
\Yellow{locally changed to yellow}. Now we are back to cyan.
```

The color commands will even work in math mode and across math mode boundaries. This means that if you have a color before going into math mode, the mathematics will be set in that color as well. More importantly however, in alignment environments like '`\halign`', '`tabular`' or '`eqnarray`', local color commands cannot extend beyond the alignment characters.

Because local color commands respect only some environment and deliminator changes besides their own, care must be taken in setting their scope. It is best not to have then stretch too far.

At the present time there are no macros for color environments in LaTeX which might have a larger range. This is primarily to keep the TeX and LaTeX use compatible.

6.2 User Definable Colors

There are two ways for the user to specify colors not already defined. For local changes, there is the command '\Color' which takes two arguments. The first argument is a quadruple of numbers between zero and one and specifies the intensity of cyan, magenta, yellow and black (CMYK) in that order. The second argument is the text that should appear in the given color. For example, suppose you want the words "this color is pretty" to appear in a color which is 50% cyan, 85% magenta, 40% yellow and 20% black. You would use the command

```
\Color{.5 .85 .4 .2}{this color is pretty}
```

For global color changes, there is a command '\textColor' which takes one argument, the CMYK quadruple of relative color intensities. For example, if you want the default color to be as above, then the command

```
\textColor{.5 .85 .4 .2}
The text from now on will be this pretty color
```

will do the trick.

Making a global color change in the midst of a nested local colors is highly discouraged. Consequently, dvips will give you warning message and do its best to recover by discarding the current color history.

6.3 Subtleties in Using Color

These color macros are defined by use of specialized '\special' keywords. As such, they are put in the '.dvi' file only as explicit message strings to the driver. The (unpleasant) result is that certain unprotected regions of the text can have unwanted color side effects. For example, if a color region is split by TeX across a page boundary, then the footers of the current page (e.g., the page number) and the headers of the next page can inherit that color. To avoid this effect globally, users should make sure that these special regions of the text are defined with their own local color commands. For example in TeX, to protect the header and footer, use

```
\headline{\Black{My Header}}
\footline{\Black{\hss\tenrm\folio\hss}}
```

This warning also applies to figures and other insertions, so be careful!

Of course, in LaTeX, this is much more difficult to do because of the complexity of the macros that control these regions. This is unfortunate, but is somehow inevitable because TeX and LaTeX were not written with color in mind.

Even when writing your own macros, much care must be taken. The color macros that 'colorize' a portion of the text work by prefixing the text with a special command to turn the color on and postfixing it with a different special command to restore the original color. It is often useful to ensure that TeX is in horizontal mode before the first special command is issued; this can be done by prefixing the color command with '`\leavevmode`'.

6.4 Printing in Black/White, after Colorizing

If you have a TeX or LaTeX document written with color macros and you want to print it in black and white there are two options. On all (good) PostScript devices, printing a color file will print in corresponding grey-levels. This is useful since in this way you can get a rough idea of where the colors are changing without using expensive color printing devices. The second option is to replace the call to input '`colordvi.tex`' with '`blackdvi.tex`' (and similarly for the '`.sty`' files). So in the above example, replacing the word '`colordvi`' with '`blackdvi`' suffices. This file defines the color macros as no-ops, and so will produce normal black/white printing. By this simple mechanism, the user can switch to all black/white printing without having to ferret out the color commands. Also, some device drivers, particularly non-PostScript ones like screen previewers, will simply ignore the color commands and so print in normal black/white. Hopefully, in the future screen previewers for color displays will be compatible with some form of color support.

6.5 Configuring dvips for Color Devices

To configure dvips for a particular color device you need to fine tune the color parameters to match your devices color rendition. To do this, you will need a PANTONE chart for your device. The header file '`color.lpro`' shows a (rough) correspondence between the Crayola crayon names and the PANTONE numbers and also defines default CMYK values for each of the colors. Note that these colors must be defined in CMYK terms and not RGB as dvips outputs PostScript color commands in CMYK. This header file also defines (if they are not known to the interpreter) the PostScript commands '`setcmykcolor`' and '`currentcmykcolor`' in terms of a RGB equivalent so if your device only understands RGB, there should be no problem.

The parameters set in this file were determined by comparing the PANTONE chart of a Tektronics PHASER printer with the actual Crayola Crayons. Because these were defined for a particular device, the actual color rendition on your device may be very different. There are two ways to adjust this. One is to use the PANTONE chart for your device to rewrite '`color.lpro`' prior to compilation and installation. A better alternative, which supports multiple devices, is to add a header file option in the configuration file for each device that defines, in '`userdict`', the color parameters for those colors that need redefining.

For example, if you need to change the parameters defining '`Goldenrod`' (approximately PANTONE 109) for your device '`mycolordev`', do the following. In the PANTONE chart for your device, find the CMYK values for PANTONE 109. Let's say they are '`{\ 0 0.10 0.75 0.03 }`'. Then create a header file named '`mycolordev.pro`' with the commands

```
userdict begin
/Goldenrod { 0 0.10 0.75 0.03 setcmykcolor} bind def
```

Finally, in '`config.mycolordev`' add the line

```
h mycolordev.pro
```

This will then define '`Goldenrod`' in your device's CMYK values in '`userdict`' which is checked before defining it in '`TeXdict`' by '`color.pro`'.

This mechanism, together with additions to '`colordvi.tex`' and '`blackdvi.tex`' (and the '`.sty`' files), can also be used to predefine other colors for your users.

6.6 Color Support Details

To support color, dvips recognizes a certain set of specials. These specials all start with the keyword '`color`' or the keyword '`background`'.

We will describe '`background`' first, since it is the simplest. The '`background`' keyword must be followed by a color specification. That color specification is used as a fill color for the background. The last '`background`' special on a page is the one that gets issued, and it gets issued at the very beginning of the page, before any text or specials are sent. (This is possible because the prescan phase of dvips notices all of the color specials so that the appropriate information can be written out during the second phase.)

Ahh, but what is a color specification? It is one of three things. First, it might be a PostScript procedure as defined in a PostScript header file. The '`color.pro`' file defines 64 of these, including

'Maroon'. This PostScript procedure must set the current color to be some value; in this case, 'Maroon' is defined as '0 0.87 0.68 0.32 setcmykcolor'.

The second possibility is the name of a color model (initially, one of 'rgb', 'hsb', 'cmyk', or 'gray') followed by the appropriate number of parameters. When dvips encounters such a macro, it sends out the parameters first, followed by the string created by prefixing 'TeXcolor' to the color model. Thus, the color specification 'rgb 0.3 0.4 0.5' would generate the PostScript code '0.3 0.4 0.5 TeXrgbcolor'. Note that the case of zero arguments is disallowed, as that is handled by the single keyword case above (where no changes to the name are made before it is sent to the PostScript file.)

The third and final type of color specification is a double quote followed by any sequence of PostScript. The double quote is stripped from the output. For instance, the color specification '"AggiePattern setpattern' will set the 'color' to the Aggie logo pattern (assuming such exists.)

So those are the three types of color specifications. The same type of specifications are used by both the 'background' special and the 'color' special. The 'color' special itself has three forms. The first is just 'color' followed by a color specification. In this case, the current global color is set to that color; the color stack must be empty when such a command is executed.

The second form is 'color push' followed by a color specification. This saves the current color on the color stack and sets the color to be that given by the color specification. This is the most common way to set a color.

The final version of the 'color' special is just 'color pop', with no color specification; this says to pop the color last pushed on the color stack from the color stack and set the current color to be that color.

The Dvips program correctly handles these color specials across pages, even when the pages are repeated or reversed.

These color specials can be used for things such as patterns or screens as well as simple colors. However, note that in the PostScript, only one 'color specification' can be active at a time. For instance, at the beginning of a page, only the bottommost entry on the color stack is sent; also, when a color is 'popped', all that is done is that the color specification from the previous stack entry is sent. No 'gsave' or 'grestore' is used. This means that you cannot easily mix usage of the 'color' specials for screens and colors, just one or the other. This may be addressed in the future by adding support for different 'categories' of color-like state.

7 Invoking Dvips

There many ways to control Dvips' behavior: configuration files, environment variables, and command-line options.

7.1 Dvips options

Dvips has a plethora of command line options. Reading through this section will give a good idea of the capabilities of the driver.

Many of the parameterless options listed here can be turned off by immediately suffixing the option with a zero (0); for instance, to turn off page reversal if it is turned on by default, use '-r0'. The options that can be turned off in this way are 'a', 'f', 'k', 'i', 'm', 'q', 'r', 's', 'E', 'F', 'K', 'M', 'N', 'U', and 'Z'.

This is a handy summary of the options; it is printed out when you run dvips with no arguments.

```
This is dvipsk version Copyright 1986, 1993 Radical Eye Software
     Usage: dvips [options] filename[.dvi]
 a*  Conserve memory, not time      y # Multiply by dvi magnification
 b # Page copies, for posters e.g.  A   Print only odd (TeX) pages
 c # Uncollated copies              B   Print only even (TeX) pages
 d # Debugging                      C # Collated copies
 e # Maxdrift value                 D # Resolution
 f*  Run as filter                  E*  Try to create EPSF
 h f Add header file                F*  Send control-D at end
 i*  Separate file per section      K*  Pull comments from inclusions
 k*  Print crop marks               M*  Don't make fonts
 l # Last page                      N*  No structured comments
 m*  Manual feed                    O c Set/change paper offset
 n # Maximum number of pages        P s Load config.$s
 o f Output file                    R   Run securely
 p # First page                     S # Max section size in pages
 q*  Run quietly                    T c Specify desired page size
 r*  Reverse order of pages         U*  Disable string param trick
 s*  Enclose output in save/restore V*  Send downloadable PS fonts as PK
 t s Paper format                   X # Horizontal resolution
 x # Override dvi magnification     Y # Vertical resolution
                                    Z*  Compress bitmap fonts
 pp #-# First-last page
     # = number   f = file   s = string  * = suffix, '0' to turn off
     c = comma-separated dimension pair (e.g., 3.2in,-32.1cm)
```

'-' Ask for additional options from standard input.

'-a' Conserve memory by making three passes over the 'dvi' file instead of two and only loading those characters actually used. Generally only useful on machines with a very limited amount of memory, like some PCs.

'-c *num*' Generate *num* copies of every page. Default is 1. (For collated copies, see the '-C' option below.)

'-b *num*' Generate *num* copies of each page, but duplicating the page body rather than using the '#numcopies' option. This can be useful in conjunction with a header file setting '\bop-hook' to do color separations or other neat tricks.

'-d *num*' Set the debug flags. This is intended only for emergencies or for unusual fact-finding expeditions; it will work only if dvips has been compiled with the 'DEBUG' option. See Section 9.1.1 [Debug Options], page 52, for the possible values of *num*. Use a value of '-1', and specify the option first, for maximum output.

'-e *num*' Make sure that each character is placed at most this many pixels from its 'true' resolution-independent position on the page. The default value of this parameter is resolution dependent (it is the number of entries in the list [100, 200, 300, 400, 500, 600, 800, 1000, 1200, 1600, 2000, 2400, 2800, 3200, ...] that are less than or equal to the resolution in dots per inch). Allowing individual characters to 'drift' from their correctly rounded positions by a few pixels, while regaining the true position at the beginning of each new word, improves the spacing of letters in words.

'-f' Run as a filter. Read the 'dvi' file from standard input and write the PostScript to standard output. The standard input must be seekable, so it cannot be a pipe. If you must use a pipe, write a shell script that copies the pipe output to a temporary file and then points dvips at this file. This option also disables the automatic reading of the 'PRINTER' environment variable; use '-P$PRINTER' after the '-f' to read it anyway. It also turns off the automatic sending of control D if it was turned on with the '-F' option or in the configuration file; use '-F' after the '-f' to send it anyway.

'-h *name*' Prepend file *name* as an additional header file. (However, if the name is simply '-', suppress all header files from the output.) This header file gets added to the PostScript 'userdict'.

'-i' Make each section be a separate file. Under certain circumstances, dvips will split the document up into 'sections' to be processed independently; this is most often done for memory reasons. Using this option tells dvips to place each section into a separate file; the new file names are created replacing the suffix of the supplied output file name by a three-digit sequence number. This option is most often used in conjunction with the '-S' option which sets the maximum section length in pages. For instance, some phototypesetters cannot print more than ten or so consecutive pages before running out of steam; these options can be used to automatically split a book into ten-page sections, each to its own file.

'-k' Print crop marks. This option increases the paper size (which should be specified, either with a paper size special or with the '-T' option) by a half inch in each dimension. It translates each page by a quarter inch and draws cross-style crop marks. It is mostly useful with typesetters that can set the page size automatically.

'-l *num*' The last page printed will be the first one numbered *num*. Default is the last page in the document. If the *num* is prefixed by an equals sign, then it (and any argument to the '-p' option) is treated as a sequence number, rather than a value to compare with '\count0' values. Thus, using '-l =9' will end with the ninth page of the document, no matter what the pages are actually numbered.

'-m' Specify manual feed for printer.

'-mode *mode*'
Use *mode* as the Metafont device name for path searching and generating fonts. This overrides any value from configuration files. With the default paths, explicitly specifying the mode also makes the program assume the fonts are in a subdirectory named *mode*. See section "TeX directory structure" in *Kpathsearch library*.

If Metafont seems not to understand the *mode* name, see Section 9.1.8 [Unable to Generate Fonts], page 55.

'-n *num*' At most *num* pages will be printed. Default is 100000.

'-o *name*' The output will be sent to file *name*. If no file name is given, the default name is '*file*.ps' where the 'dvi' file was called '*file*.dvi'; if this option isn't given, any default in the configuration file is used. If the first character of the supplied output file name is '!' or '|', then the remainder will be used as an argument to 'popen'; thus, specifying '|lpr' as the output file will automatically queue the file for printing as usual. This option also disables the automatic reading of the 'PRINTER' environment variable, and turns off the automatic sending of control D. See the '-f' option for how to override this.

'-p *num*' The first page printed will be the first one numbered *num*. Default is the first page in the document. If the *num* is prefixed by an equals sign, then it (and any argument to the '-l' option) is treated as a sequence number, rather than a value to compare with '\count0' values. Thus, using '-p =3' will start with the third page of the document, no matter what the pages are actually numbered.

'-pp *first-last*'
Print pages *first* through *last*; equivalent to '-p *first* -l *last*'. The '-' range separator can also be a ':'.

'-q' Run quietly. Don't chatter about pages converted, etc.; report nothing but errors to standard error.

'-r' Stack pages in reverse order. Normally, page 1 will be printed first.

'-s' Causes the entire global output to be enclosed in a save/restore pair. This causes the file to not be truly conformant, and is thus not recommended, but is useful if you are driving the printer directly and don't care too much about the portability of the output.

'-t *papertype*'
This sets the paper type to *papertype*. The *papertype* should be defined in one of the configuration files, along with the appropriate code to select it. See the documentation for '@'in the configuration file option descriptions. You can also specify '-t landscape', which rotates a document by 90 degrees. To rotate a document whose size is not letter, you can use the '-t' option twice, once for the page size, and once for 'landscape'. The upper left corner of each page in the 'dvi' file is placed one inch from the left and one inch from the top. Use of this option is highly dependent on the configuration file. Note that executing the 'letter' or 'a4' or other PostScript operators cause the document to be nonconforming and can cause it not to print on certain printers, so the default paper size should not execute such an operator if at all possible.

'-x *num*' Set the magnification ratio to *num*/1000. Overrides the magnification specified in the 'dvi' file. Must be between 10 and 100000. It is recommended that you use standard magstep values (1095, 1200, 1440, 1728, 2074, 2488, 2986, and so on) to help reduce the total number of PK files generated.

'-A' This option prints only the odd pages. This option uses the \TeX\ page numbering rather than the sequence page numbers.

'-B' This option prints only the even pages. This option uses the \TeX\ page numbering rather than the sequence page numbers.

'-C *num*' Create *num* copies, but collated (by replicating the data in the PostScript file). Slower than the '-c' option, but easier on the hands, and faster than resubmitting the same PostScript file multiple times.

'-D *num*' Set the resolution in dpi (dots per inch) to *num*. This affects the choice of bitmap fonts that are loaded and also the positioning of letters in resident PostScript fonts. Must be between 10 and 10000. This affects both the horizontal and vertical resolution. If a high resolution (something greater than 400 dpi, say) is selected, the '-Z' flag should probably also be used.

'-E' Makes dvips attempt to generate an EPSF file with a tight bounding box. This only works on one-page files, and it only looks at marks made by characters and rules, not by any included graphics. In addition, it gets the glyph metrics from the 'tfm' file, so characters that lie outside their enclosing 'tfm' box may confuse it. In addition, the bounding box might be a bit too loose if the character glyph has significant left or right side bearings. Nonetheless, this option works well for creating small EPSF files for equations or tables or the like. (Of course, Dvips output is resolution dependent and thus does not make very good EPSF files, especially if the images are to be scaled; use these EPSF files with a great deal of care.)

'-F' Causes Control-D (ASCII code 4) to be appended as the very last character of the PostScript file. This is useful when dvips is driving the printer directly instead of working through a spooler, as is common on extremely small systems. Otherwise, it is not recommended.

'-K' This option causes comments in included PostScript graphics, font files, and headers to be removed. This is sometimes necessary to get around bugs in spoolers or PostScript post-processing programs. Specifically, the '%%Page' comments, when left in, often cause difficulties. Use of this flag can cause some included graphics to fail, since the PostScript header macros from some software packages read portions of the input stream line by line, searching for a particular comment. This option has been turned off by default because PostScript previewers and spoolers have been getting better.

'-M' Turns off the automatic font generation facility. If any fonts are missing, commands to generate the fonts are appended to the file 'missfont.log' in the current directory; this file can then be executed to create the missing fonts.

'-N' Turns off structured comments; this might be necessary on some systems that try to interpret PostScript comments in weird ways, or on some PostScript printers. Old versions of TranScript in particular cannot handle modern Encapsulated PostScript.

'-O *offset*' Move the origin by a certain amount. The *offset* is a comma-separated pair of dimensions, such as '.1in,-.3cm' (in the same syntax used in the 'papersize' special). The origin of the page is shifted from the default position (of one inch down, one inch to the right from the upper left corner of the paper) by this amount.

'-P *printername*'
Sets up the output for the appropriate printer. This is implemented by reading in 'config.*printername*', which can then set the output pipe (as in, 'o !lpr -Pprintername') as well as the font paths and any other defaults for that printer only. It is recommended that all standard defaults go in the one master 'config.ps' file and only things that vary printer to printer go in the 'config.*printername*' files. Note that 'config.ps' is read before 'config.*printername*'. In addition, another file called '~/.dvipsrc' is searched for immediately after 'config.ps'; this file is intended for user defaults. If no '-P' command is given, the environment variable 'PRINTER' is checked. If that variable exists, and a corresponding configuration file exists, that configuration file is read in.

'-R' Run securely. This disables command execution in '\special' (via '`') and config files (via the 'E' option), pipes as output files, and opening of any absolute filenames.

'-S *num*' Set the maximum number of pages in each 'section'. This option is most commonly used with the '-i' option; see that documentation above for more information.

'-T *offset*' Set the paper size to the given pair of dimensions. This option takes its arguments in the same style as '-O'. It overrides any paper size special in the 'dvi' file.

'-U' Disable a PostScript virtual memory saving optimization that stores the character metric information in the same string that is used to store the bitmap information. This is only necessary when driving the Xerox 4045 PostScript interpreter. It is caused by a bug in that interpreter that results in 'garbage' on the bottom of each character. Not recommended unless you must drive this printer.

'-V' Download non-resident PostScript fonts as bitmaps. This requires use of `mtpk` or `pstopk` or some combination of the two in order to generate the required bitmap fonts; neither of these programs are supplied with Dvips.

'-X *num*' Set the horizontal resolution in dots per inch to *num*.

'-Y *num*' Set the vertical resolution in dots per inch to *num*.

'-Z' Causes bitmapped fonts to be compressed before they are downloaded, thereby reducing the size of the PostScript font-downloading information. Especially useful at high resolutions or when very large fonts are used. Will slow down printing somewhat, especially on early 68000-based PostScript printers.

7.2 Environment Variables

Dvips reads a certain set of environment variables to configure its operation. The path variables are read as needed, after all configuration files are read, so they override values in the configuration files. (Except for the 'TEXCONFIG' variable.)

See section "Path specifications" in *Kpathsea library*, for details of interpretation of environment variable values, and the common environment variables. Only the environment specific to Dvips are mentioned here.

'DVIPSFONTS'
: Default path to search for all fonts. Overrides all the font-related config file options and other environment variables.

'DVIPSHEADERS'
: Default path to search for PostScript header files. Overrides the 'H' config file option.

'DVIPSMAKEPK'
: Overrides 'MakeTeXPK' as the name of the program to invoke to create missing PK fonts. You can change the arguments passed to the 'MakeTeXPK' program with the 'MAKETEXPK' environment variables; see the Kpathsea documentation.

'DVIPSSIZES'
: Last-resort sizes for scaling for unfound fonts. Overrides the 'R' definition in config files.

'PRINTER'
: This environment variable is read to determine which default printer configuration file to read in. It is the responsibility of the configuration file to send output to the proper print queue, if such functionality is desired.

'TEXCONFIG'
: This environment variable sets the directories to search for configuration files, including the system-wide one. Using this single environment variable and the appropriate configuration files, it is possible to set up the program for any environment. (The other path environment variables are thus redundant.)

'TEXPICTS'
: Path to search for special illustrations. Overrides the 'S' config file option. If not set, 'TEXINPUTS' is used.

7.3 Configuration Files

This section describes the configuration files specific to Dvips. The driver also reads the configuration files generic to Kpathsea (see section "Config files" in *Kpathsearch library*).

7.3.1 Configuration File Searching

The dvips program loads many different configuration files, so that parameters can be set globally across the system, on a per-printer basis, or individually by each user.

When dvips starts up, first the global 'config.ps' file is searched for and loaded. This file is looked for along the path for configuration files, which is set in the 'Makefile' at compilation. After this master configuration file is loaded, a file by the name of '.dvipsrc' is loaded from the current user's home directory, if such a file exists. This file is loaded in exactly the same way as the global configuration file, and it can override any options set in the global file.

Then the command line is read and parsed. If the '-P' option is encountered, at that point in the command line a configuration file for that printer is read in. Thus, the printer configuration file can override anything in the global or user configuration file, and it can also override most options in the command line up to the point that the '-P' option was encountered.

After the command line has been completely scanned, if there was no '-P' option selected, and also the '-o' and '-f' command line options were not used, a 'PRINTER' environment variable is searched for. If this variable exists, and a configuration file for the printer mentioned in it exists, this configuration file is loaded last of all.

Because the printer-specific configuration files are read after the user's configuration file, the user cannot override things in the printer configuration files. On the other hand, the configuration path usually includes the current directory, and can be set to include the user's home directory (or any other directory of the user), so the user can always provide his or her own printer-specific configuration files that will be found before the system global ones.

A few options are treated specially, in that they are not overridden by configuration files:

'-mode'
: This overrides any mode setting ('M' line) in 'config.*'. '-mode' does not affect the resolution, however.

'-o'
: This overrides any output setting ('o' line) in 'config.*'.

'-D'
: As well as setting the current resolution, this unsets the mode if the mode was set from a configuration file. If 'config.$PRINTER' is read, however, any 'D' or 'M' lines from it will take effect.

The purpose of these special cases is to (1) minimize the chance of having a mismatched mode and resolution (which 'MakeTeXPK' cannot handle), and (2) to let command-line options override config files.

7.3.2 Configuration File Options

Most of the configuration file options are similar to the command line options, but there are a few differences. As with command line options, many may be turned off by suffixing the letter with a zero (0). These options are 'afqrKNUZ'.

Within a configuration file, any empty line or line starting with a space, asterisk, equal sign, or a pound sign is ignored.

'`@` *name hsize vsize*'

This option is used to set the paper size defaults and options for the particular printer this configuration file describes. There are three formats for this option. If the option is specified on a line by itself, with no parameters, it instructs dvips to discard all other paper size information (possibly from another configuration file) and start fresh. If three parameters are given, as above, with the first parameter being a name and the second and third being a dimension (as in 8.5in or 3.2cc, just like in the '`papersize`' special), then the option is interpreted as starting a new paper size description, where *name* is the name and *hsize* and *vsize* describe the horizontal and vertical size of the sheet of paper, respectively. If both *hsize* and *vsize* are equal to zero (although you must still specify units!) then any page size will match it. If the '`@`' character is immediately followed by a '`+`' sign, then the remainder of the line (after skipping any leading blanks) is treated as PostScript code to send to the printer to select that particular paper size. After all that, if the first character of the line is an exclamation point, then the line is put in the initial comments section of the final output file; else, it is put in the setup section of the output file. For instance, a subset of the paper size information supplied in the default '`config.ps`' looks like

```
@ letterSize 8.5in 11in
@ letter 8.5in 11in
@+ %%BeginPaperSize: Letter
@+ letter
@+ %%EndPaperSize
@ legal 8.5in 14in
@+ ! %%DocumentPaperSizes: Legal
@+ %%BeginPaperSize: Legal
@+ legal
@+ %%EndPaperSize
```

Note that you can even include structured comments in the configuration file! When dvips has a paper format name given on the command line, it looks for a match by the *name*; when it has a '`papersize`' special, it looks for a match by dimensions. The first match found (in the order the paper size information is found in the configuration file) is used. If nothing matches, a warning is printed and the first paper size given is used, so the first paper size should always be the default. The dimensions must match within a quarter of an inch. Landscape mode for all of the paper sizes are automatically supported. If your printer has a command to set a special paper size, then give dimensions of '`0in 0in`'; the PostScript code that sets the paper size can refer to the dimensions the user requested as '`hsize`' and '`vsize`'; these will be macros defined in the PostScript that return the requested size in default PostScript units. Note that virtually all of the PostScript commands you use here are device dependent and degrade the portability of the file; that is why the default first paper size entry should not send any PostScript commands down (although a structured comment or two would be okay). Also, some printers want '`BeginPaperSize`' comments and paper size setting commands; others (such as the NeXT) want '`PaperSize`' comments and they will handle setting the paper size. There is no solution I could find that works for both (except maybe specifying both.) See the supplied '`config.ps`' file for a more realistic example.

'`a`' Conserve memory by making three passes over the '`dvi`' file instead of two and only loading those characters actually used. Generally only useful on machines with a very limited amount of memory, like some PCs.

'`b` *num*' Generate *num* copies of each page, but duplicating the page body rather than using the '`#numcopies`' option. This can be useful in conjunction with a header file setting '`\bop-hook`' to do color separations or other neat tricks.

'`e` *num*' Set the maximum drift parameter to *num* dots (pixels) as explained above.

'`f`' Run as a filter by default.

'`h` *name*' Add *name* as a PostScript header file to be downloaded at the beginning.

'`i` *num*' Make each section be a separate file, and set the maximum number of pages in a given file to *num*. Under certain circumstances, dvips will split the document into 'sections' to be processed independently; this is most often done for memory reasons. Using this option tells dvips to place each section into a separate file; the new file names are created by replacing the suffix of the supplied output file name with a three-digit sequence number. This is essentially a combination of the command line options '`-i`' and '`-S`'; see the documentation for these options for more information.

'`m` *num*' The value *num* is the virtual memory available for fonts and strings in the printer. Default is 180000. This value must be accurate if memory conservation and document splitting is to work correctly. To determine this value, send the following file to the printer:

```
%! Hey, we're PostScript
/Times-Roman findfont 30 scalefont setfont 144 432 moveto
vmstatus exch sub 40 string cvs show pop showpage
```

Note that the number returned by this file is the total memory free; it is often a good idea to tell dvips that the printer has somewhat less memory. This is because many programs download permanent macros that can reduce the memory in the printer. In general, a memory size of about '`300000`' is plenty, since dvips can automatically split a document if required. It is unfortunate that PostScript printers with much less virtual memory still exist. Some systems or printers can dynamically increase the memory available to a PostScript interpreter, in which case this file might return a ridiculously low number; the NeXT computer is such a machine. For these systems, a value of one million works well.

'`o` *name*' The default output file is set to *name*. As above, it can be a pipe. Useful in printer-specific configuration files to redirect the output to a particular printer queue.

'`p` *name*' The file to examine for PostScript font aliases is *name*. It defaults to '`psfonts.map`'. This option allows different printers to use different resident fonts. If the name starts with a '`+`' character, then the rest of the name (after any leading spaces) is used as an additional map file; thus, it is possible to have local map files pointed to by local configuration files that append to the global map file.

'`q`' Run quietly by default.

'`r`' Reverse the order of pages by default.

'`s`' Enclose the entire document in a global save/restore pair by default. Not recommended, but useful in some environments; this breaks the conformance of the document to the Adobe PostScript structuring conventions.

'`D` *num*' Set the vertical and horizontal resolution to *num* dots per inch. Useful in printer-specific configuration files.

'`E` *command*'
Executes the command listed; can be used to get the current date into a header file for inclusion, for instance. Possibly dangerous; in many installations this may be disabled, in which case a warning message will be printed if the option is used.

'`H` *path*' The (colon-separated) path to search for PostScript header files is *path*. The environment variable '`DVIPSHEADERS`' overrides this.

'`K`' Filter comments out of included PostScript files; see the description above for more information.

'`M` *mode*' Set *mode* as the METAFONT mode to be used when generating fonts and for path searching. The '`-mode`' option overrides this. With the default paths, specifying a mode this way also makes the program assume the fonts are in a subdirectory named *mode*. See section "TeX directory structure" in *Kpathsearch library*.

'`N`' Disable PostScript comments by default.

'`O` *offset*' Move the origin by a certain amount. The *offset* is a comma-separated pair of dimensions, such as '`.1in,-.3cm`' (in the same syntax as used in the '`papersize`' special). The origin of the page is shifted from the default position (of one inch down, one inch to the right from the upper left corner of the paper) by this amount.

This is useful for a printer that consistently offsets output pages by a certain amount. You can use the file '`testpage.tex`' to determine the correct value for your printer. Be sure to do several runs with the same `O` value—some printers vary widely from run to run.

'`P` *path*' The (colon-separated) path to search for bitmap '`pk`' font files is *path*. The '`PKFONTS`', '`TEXPKS`', '`GLYPHFONTS`', and '`TEXFONTS`' environment variables override this. See section "TeX environment variables" in *Kpathsearch library*.

'R *num num* ...'
: Sets up a list of default resolutions to search for 'pk' fonts, if the requested size is not available. The output will then scale the font found using PostScript scaling to the requested size. The resulting output may be ugly, and thus a warning is issued. To turn this off, use a line with just the 'R' and no numbers.

'S *path*'
: The path to search for special illustrations (Encapsulated PostScript files or psfiles) is *path*. The 'TEXPICTS' and then 'TEXINPUTS' environment variables override this.

'T *path*'
: The path to search for the 'tfm' files is *path*. The 'TFMFONTS' and then 'TEXFONTS' environment variables overrides this. This path is used for resident fonts and fonts that can't otherwise be found.

'U'
: Turns off a memory-saving optimization; this is necessary for the Xerox 4045 printer, but not recommended otherwise. See the description above for more information.

'V *path*'
: The path to search for virtual font 'vf' files is *path*. This may be device-dependent if you use virtual fonts to simulate actual fonts on different devices.

'W [*string*]'
: If *string* is supplied, write it to stderr after reading all the configuration files; otherwise, cancel any previous 'W' message. This is useful in the default configuration file if you want to require the user to specify a printer, for instance, or if you want to notify the user about special characteristics of the resultant output from a particular printer.

'X *num*'
: Set the horizontal resolution to *num* dots per inch.

'Y *num*'
: Set the vertical resolution to *num* dots per inch.

'Z'
: Compress all downloaded fonts by default, as above.

8 Other Bells And Whistles

For special effects, if any of the macros '`bop-hook`', '`eop-hook`', '`start-hook`', or '`end-hook`' are defined in the PostScript '`userdict`', they will be executed at the beginning of a page, end of a page, start of the document, and end of a document, respectively.

When these macros are executed, the default PostScript coordinate system and origin is in effect. Such macros can be defined in headers added by the '`-h`' option or the '`header=`' special, and might be useful for writing, for instance, DRAFT across the entire page, or, with the aid of a shell script, dating the document. These macros are executed outside of the save/restore context of the individual pages, so it is possible for them to accumulate information, but if a document must be divided into sections because of memory constraints, such added information will be lost across section breaks.

The single argument to '`bop-hook`' is the sequence number of the page in the file; the first page gets zero, the second one, etc. The procedure must leave this number on the stack. None of the other hooks are (currently) given parameters, although this may change in the future.

As an example of what can be done, the following special will write a light DRAFT across each page in the document:

```
\special{!userdict begin /bop-hook{gsave 200 30 translate
65 rotate /Times-Roman findfont 216 scalefont setfont
0 0 moveto 0.7 setgray (DRAFT) show grestore}def end}
```

Note that using '`bop-hook`' or '`eop-hook`' in any way that preserves information across pages will break compliance with the Adobe document structuring conventions, so if you use any such tricks, it is recommended that you also use the '`-N`' option to turn off structured comments.

Several of the above tricks can be used nicely together, and it is not necessary that a 'printer configuration file' be used only for printer. For instance, a '`-P`' file can be set up to print the date on each page; the particular configuration file will execute a command to put the date into a header file, which is then included with a '`h`' line in the configuration file. Multiple '`-P`' options can be used.

9 Installation

To compile and install Dvipsk:

1. Edit the file '`make/paths.make`' if you want to make changes to the installation directories or paths that will have effect across different runs of `configure`. Alternatively, override the Make variables on the command line when you run Make.

   Exception: to reliably change the top-level '`prefix`', you must give '`configure`' the option '`-prefix=`*prefix*', instead of changing the value in '`paths.make`'.

2. Edit '`kpathsea/texmf.cnf.in`' to change the local paths to match your local setup. See section "Default paths" in *Kpathsea*, for more details on changing the paths. A copy is in '`kpathsea/INSTALL`'. See '`kpathsea/HIER`' for an explanation of the default setup.

   If the paths do not match where the files actually are, the programs will probably start up Very, Very, Slowly, and/or not be able to find the fonts or other input files.

3. `sh configure` (in the top-level directory). This makes system-dependent '`#define's`' in '`*/c-auto.h`' (from the corresponding '`c-auto.h.in`') and creates a '`Makefile`' (from the corresponding '`Makefile.in`', by doing '`@`*var*`@`' and '`ac_include`' substitutions).

   Perhaps the most common desire is to compile with optimization instead of or as well as debugging. You can change the options passed to the compiler by changing '`CFLAGS`', either for `configure` or `make`. For example:

   ```
   prompt$ env CFLAGS="-g -O" configure
   prompt$ make
   ```

   or

   ```
   prompt$ configure
   prompt$ make CFLAGS="-g -O"
   ```

   See section "Running `configure` scripts" in *Autoconf*, for detailed `configure` options. (A copy is in '`kpathsea/CONFIGURE`'.)

4. `make` (still in the top-level directory). Barring configuration and compiler bugs, this will compile all the programs. See section "Common problems" in *Kpathsea*, for system-dependent problems (this section is also in '`kpathsea/INSTALL`').

   This also creates the '`texmf.cnf`' and '`paths.h`' files that define the default search paths.

1. Check the paths in '`MakeTeXPK`', unless you do not want automatic font generation. See Section 9.2 [Font Generation], page 55. The '`MakeTeXPK`' in the distribution will overwrite the installed file only if the latter contains the string '`original MakeTeXPK --`'.

   Dvipsk, unlike the original dvips, *requires* '`MakeTeXPK`' to echo the generated filename (and nothing else) to standard output (standard error can be used for commentary). For more details, or in general if '`MakeTeXPK`' fails, see Section 9.1.8 [Unable to Generate Fonts], page 55.

By default, '`MakeTeXPK`' installs the new PK fonts under '`/usr/local/lib/texmf/fonts/tmp/pk`'. For the simplest installation, create that directory and make it publically writable. See Section 9.2 [Font Generation], page 55, for alternatives.

2. Update the device parameters (available memory, resolution, etc.) in '`config.ps`'. This file is installed as the system-wide configuration file. See Section 7.3.2 [Config File Options], page 44. The '`config.ps`' in the distribution will overwrite the installed file only if the latter contains the string '`original config.ps --`'.

   If you need support for more than one device, create configuration files for each and install them in the directory named by the Make variable `configdir`. See the '`-P`' option in Chapter 7 [Invoking Dvips], page 37.

3. Install the programs and supporting macros, fonts, and data files with `make install`. If you want to install only the executables, do `make install-exec`; for only the data files, `make install-data`. And if you don't want to install the fonts (perhaps because your directory structure is different from the default), but do want everything else, set the Make variable `install_fonts=false`.

4. Install additional fonts, if you want to.

   A few Type 1 fonts (Utopia, Charter, Courier, Nimbus, Antiqua, ...) have been contributed to the X Consortium, and thus are freely available. You can get TeX distributions for them from '`ftp.cs.umb.edu`' in '`pub/tex`', and from the CTAN hosts in '`tex-archive/fonts`'.

   If you have a commercial Unix system, it may have come with additional PostScript fonts. If so, you can make them available to Dvips by (1) copying or linking them with the appropriate filenames; and (2) running Afm2tfm (see ⟨undefined⟩ [Installing PostScript fonts], page ⟨undefined⟩) to make TFM and VF files so the fonts will be available in the same encoding as the fonts distributed with Dvips. Also check '`psfonts.map`' to be sure the fonts are listed there (see Section 5.4 [psfonts.map], page 29).

   Here are the typical locations for vendor-supplied fonts:

   DEC Ultrix
   : '`/usr/lib/DPS/outline/decwin`'

   DEC OSF/1
   : '`/usr/lib/X11/fonts/Type1Adobe`'

   IBM AIX
   : '`/usr/lpp/DPS/fonts/outlines`'

   NeXT
   : '`/NextLibrary/Fonts/outline`'

   SGI IRIX
   : '`/usr/lib/DPS/outline/base`'

   Sun Solaris
   : '`/usr/openwin/lib/X11/fonts/Type1/outline`'

   The NeXT system supplies more fonts than the others, but the sets are overlapping. See the distributed '`psfonts.map`' for which fonts each system supplies.

5. `make distclean`. This removes all files created by the build.

See Section 9.1.1 [Debug Options], page 52, for runtime debugging options that may help track down problems.

See section "Reporting bugs" in *Kpathsea*, for the bug reporting address and information. (Also at the end of '`kpathsea/INSTALL`'.)

9.1 Diagnosing problems

You've gone through all the trouble of installing dvips, carefully read all the instructions in this manual, and still can't get something to work. This is all too common, and is usually caused by some broken PostScript application out there. The following sections provide some helpful hints if you find yourself in such a situation.

In all cases, you should attempt to find the smallest file that causes the problem. This will not only make debugging easier, it will also reduce the number of possible interactions among different parts of the system.

9.1.1 Debug Options

The '`-d`' flag to dvips is very useful for helping to track down certain errors. The parameter to this flag is an integer that tells what errors are currently being tracked. To track a certain class of debug messages, simply provide the appropriate number given below; if you wish to track multiple classes, sum the numbers of the classes you wish to track. To track all classes, you can use `-1` (output is extremely voluminous). Another useful value is `3650`, which tracks everything having to do with file searching and opening.

The classes are:

1 specials

2 paths

4 fonts

8 pages

16 headers

32	font compression
64	files
128	memory
256	Kpathsea `stat` calls
512	Kpathsea hash table lookups
1024	Kpathsea path element expansion
2048	Kpathsea path searches

9.1.2 No Output At All

If you are not getting any output at all, even from the simplest one-character file (for instance, '`\ \bye`'), then something is very wrong. Practically any file sent to a PostScript laser printer should generate some output, at the very least a page detailing what error occurred, if any. Talk to your system administrator about downloading a PostScript error handler. (Adobe distributes a good one called '`ehandler.ps`'.)

It is possible, especially if you are using non-Adobe PostScript, that your PostScript interpreter is broken. Even then it should generate an error message. I've tried to work around as many bugs as possible in common non-Adobe PostScript interpreters, but I'm sure I've missed a few.

If dvips gives any strange error messages, or compilation on your machine generated a lot of warnings, perhaps the dvips program itself is broken. Carefully check the types in '`dvips.h`' and the declarations in the '`Makefile`', and try using the debug options to determine where the error occurred.

It is possible your spooler is broken and is misinterpreting the structured comments. Try the '`-N`' flag to turn off structured comments and see what happens.

9.1.3 Output Too Small or Inverted

If some documents come out inverted or too small, your spooler is not supplying an end of job indicator at the end of each file. (This happens a lot on small machines that don't have spoolers.) You can force dvips to do this with the '`-F`' flag, but note that this generates files with a binary character (control-D) in them. You can also try using the '`-s`' flag to enclose the entire job in a save/restore pair.

9.1.4 Error Messages From Printer

If your printer returns error messages, the error message gives very good information on what might be going wrong. One of the most common error messages is '`bop undefined`'. This is caused by old versions of Transcript and other spoolers that do not properly parse the setup section of the PostScript. To fix this, turn off structured comments with the '`-N`' option, but make sure you get your spooling software updated.

Another error message is '`VM exhausted`'. (Some printers indicate this error by locking up; others quietly reset.) This is caused by telling dvips that the printer has more memory than it actually does, and then printing a complicated document. To fix this, try lowering the parameter to '`m`' in the configuration file; use the debug option to make sure you adjust the correct file.

Other errors may indicate that the graphics you are trying to include don't nest properly in other PostScript documents, or any of a number of other possibilities. Try the output on a QMS PS-810 or other Adobe PostScript printer; it might be a problem with the printer itself.

9.1.5 400 DPI Is Used Instead Of 300 DPI

This common error is caused by not editing the '`config.ps`' file to reflect the correct resolution for your site. You can use the debug flags ('`-d64`') to see what files are actually being read.

9.1.6 Long Documents Fail To Print

This is usually caused by incorrectly specifying the amount of memory the printer has in '`config.ps`'; see the description above.

9.1.7 Including Graphics Fails

The reasons why graphics inclusions fail are too numerous to mention. The most common problem is an incorrect bounding box; read the section on bounding boxes and check your PostScript file. Complain very loudly to whoever wrote the software that generated the file if the bounding box is indeed incorrect.

Another possible problem is that the figure you are trying to include does not nest properly; there are certain rules PostScript applications should follow when generating files to be included.

The dvips program includes work-arounds for such errors in Adobe Illustrator and other programs, but there are certainly applications that haven't been tested.

One possible thing to try is the '`-K`' flag, to strip the comments from an included figure. This might be necessary if the PostScript spooling software does not read the structuring comments correctly. Use of this flag will break graphics from some applications, though, since some applications read the PostScript file from the input stream looking for a particular comment.

Any application which generates graphics output containing raw binary (not ASCII hex) will probably fail with dvips.

9.1.8 Unable to Generate Fonts

This happens a lot if either '`MakeTeXPK`' hasn't been properly edited and installed, or if the local installation of METAFONT isn't correct.

'`MakeTeXPK`' must echo the generated filename (and nothing else) to standard output. See Section 9.2 [Font Generation], page 55.

If METAFONT isn't found when '`MakeTeXPK`' is running, then you need to install it. Retrieve it from, e.g., '`ftp.cs.umb.edu`' in '`pub/tex/web2c.tar.gz`' and '`pub/tex/web.tar.gz`'.

If METAFONT runs but generates fonts that are too large (and prints out the name of each character as well as just a character number), then your METAFONT base file probably hasn't been made properly. To make a proper '`plain.base`', assuming the local mode definitions are contained in '`modes.mf`' type the following command (assuming UNIX):

```
inimf "plain; input modes; dump"
```

Then copy the '`plain.base`' file from the current directory to where the base files are stored on your system.

9.2 Automatic Font Generation

One major problem with TeX and the Computer Modern fonts is the huge amount of disk space a full set of high resolution fonts can take. Dvips solves this problem by creating fonts on demand, so only those fonts that are actually used are stored on disk. At a typical site, less than one-fifth of the full set of Computer Modern fonts are used over a long period, so this saves a great deal of disk space.

Furthermore, the addition of dynamic font generation allows fonts to be used at any size, including typesetter resolutions and extremely huge banner sizes. Nothing special needs to be done; the fonts will be automatically created and installed as needed.

The downside is that it does take a certain amount of time to create a new font if it has never been used before. But once a font is created, it will exist on disk, and the next time that document is printed it will print very quickly.

It is the '`MakeTeXPK`' shell script that is responsible for making these fonts. It *must* echo the filename of the new font (and nothing else) to standard output. Use standard error for commentary.

'`MakeTeXPK`' is passed various arguments, the first of which is the font it's supposed to make. You can override the other argument conventions with environment variables or at compilation time; see the Kpathsea documentation.

The '`MakeTeXPK`' script supplied invokes Metafont (using the Sauter scripts first, if necessary and if they are installed) to create the font and then copies the resultant '`pk`' file to a world-writable font cache area.

'`MakeTeXPK`' can be customized to do other things to get the font. For instance, if you are installing dvips to replace (or run alongside) an existing PostScript driver, and that driver demands '`gf`' fonts, you can easily modify '`MakeTeXPK`' to invoke '`gftopk`' to convert the '`gf`' files to '`pk`' files for dvips. This provides the same space savings listed above.

Because dvips (and thus '`MakeTeXPK`') is run by a wide variety of users, there must be a system-wide place to put the cached font files. In order for everyone to be able to supply fonts, the directory must be world writable. If your system administrator considers this a security hole, '`MakeTeXPK`' can write to '`/tmp/pk`' or some such directory, and periodically the cached fonts can be moved to a more general system area. The cache directory must exist on the '`pk`' file search path in order for '`MakeTeXPK`' to work.

Index

M

N

O

P

Q

R

S

T

U

V

W

Short Contents

Table of Contents