1 Introduction

For those reading this on-line, the table of contents starts on page -3; the index starts on page 109.

Welcome to the world of TEX on the NeXT machine. If you have used TEX on other computer systems, you will feel at home with NeXTTEX. If you are a newcomer to TEX, welcome! You will find the NeXT machine to be a powerful computer on which to learn TEX and develop your documents.

A few words on how to read and use this manual: There are many tradeoffs to be considered in designing a TEX package; the approach taken with NeXTTEX was to create a package which is extremely powerful and highly flexible. Advanced TEX users who are comfortable with computers will find the NeXTTEX package supports countless options and is highly configurable; however, all of these options can prove very confusing to the novice TEX user or to someone unfamiliar with the basics of Unix workstations. For these users NeXTTEX assumes default values which, in a standard installation, make its use straightforward.

This manual details virtually every aspect of NeXTTEX (for those expert enough to make use of them) but for the beginner, it is sufficient (and probably preferable) to start by reading only Chapters 1, 2 and 4 (a list of additional references is supplied in Appendix A, and Appendix G gives a glossary of commonly used terms). As you learn more about both TEX and the NeXTTEX package it should prove worthwhile to return to those aspects of the manual which deal with the more advanced options. Thus, keep the manual around and refer to it often.

This document does not attempt to teach either TEX or METAFONT. For that, you should pick up one of the books listed in Appendix A of this manual. The languages these programs use is identical to any other implementation of TEX or METAFONT, by necessity; nothing has been omitted. Of course, the multitasking operating system and graphic capabilities of the NeXT machine can make typography much more enjoyable than it is on other machines.

We want to hear from you! Positive comments inspire further work on the product, and negative comments indicate a direction to proceed. If you have any difficulty getting the package up, let us know! Please write to Radical Eye Software at Box 2081, Stanford, CA 94309. A technical support BBS is available at (415) 32-RADIO, or we can be reached as `radical.eye` on BIX. We also have a technical support conference on BIX, also under

the name of `radical.eye`. Our FAX number is (415) 327-3329. Do not let the complexity of the package intimidate you; if you can't get something to work, either we haven't explained things correctly, or there is a bug. Before we jump in, a few comments must be made about conventions used in this manual.

All commands and file names will be set in typewriter type. Commands are separated from the accompanying text by blank lines, and are preceded by any prompts that are not typed by the user. For instance, to list the size of the file `foo.bar`, you would type

```
nexthost> ls -l foo.bar
```

In this example, '`nexthost>` ' is the shell prompt (yours may be different), and you would not type those letters. The word `tex` refers to the program TEX as it has been ported to the NeXT machine; the word TEX refers to the more general software package and typesetting language that runs on a wide variety of machines. Any unfamiliar terms may be looked up in the glossary in Appendix G.

1. First Run

Before we get into details of the TEX environment on the NeXT machine, we will process and print a sample document. But first, TEX must be installed on the machine if it doesn't already exist.

1.1 Installing TEX

If TEX is not installed on your system, you cannot run it. With the browser, look for `/NextDeveloper/Demos/TeXview.app`. If this file does not exist, you need to install TEX. If it does exist, you can skip the remainder of this section.

You must be superuser to install TEX. Log out of the Workspace and log back in as root. Run the `Installer` application (it's usually hiding in /NextAdmin) and select `open` in its menus. With the resulting open panel file browser, find the `NeXTTeX.pkg` file. This is normally in `/NextCD/Packages` on the CD-ROM that 3.0 came on. Select this file.

The `Installer` will then take over. It will complain that certain files will be overwritten; this is normal. It will also mention that it has to run some programs to install it; this is also okay. The entire installation process will take approximately ten minutes and will require about six megabytes of hard disk space.

Once the `Installer` completes, you can hide or quit it. You should then log out of the Workspace and log back in as a normal user; please do not ever run NeXTTEX as superuser or root.

1.2 Running TEX

Now you are ready to run TEX. If you are ambitious and familiar with TEX, you may choose to prepare a short sample file at this time. For those not familiar with TEX and for those in a hurry, we have prepared a few sample files. To run TEX over a file, just double-click on a `.tex` file in the Workspace browser; the `TeXview` application will automatically be launched and will start processing the file.

For example, with your favorite text editor, create a new file called `foo.tex` that looks exactly like

```
This is a sample \TeX\ file!\bye
```

Make sure you save this file as ASCII and not `rtf` or any other format. (In the `Edit` application, you must select the `Format/Text/Make ASCII` menu entry to edit in ASCII mode.)

After saving the file under the name `foo.tex`, double-click the `foo.tex` file in the Workspace browser. `TeXview` should be launched and a window will appear displaying something like

```
- tex -v "foo"
This is CTeX, NeXT Version 3.141 (preloaded format=plain 92.3.21)
(foo.tex [1] )
Output written on foo.dvi (1 page, 248 bytes).
Transcript written on foo.log.
```

After that appears, a larger window will appear with a preview of your output.

If you are ambitious, you might try figuring out how to work the TEXview program at this point, using what you know about gadgets and menus. Otherwise, you can exit the program by selecting the `quit` option in the menu.

1.3 Printing a Document

If you have a printer attached to your NeXT machine or network, you can easily print your sample file. Simply select the `Print` menu option of TEXview, select any desired options, and hit the `Print` button. After a second or two your document should come out of the printer. If it doesn't, inform the system administrator. If you are maintaining the system yourself, don't fret; more information is given later on installation and support.

2. What Is New for 3.0?

The 3.0 release of NeXTTEX has many improvements over the 2.0 release. TEX, `dvips`, TEXview, and many other parts of the system have been substantially revised. More details on all of these changes are given later in this manual.

2.1 Changes to TEX

TEX now supports character remapping, so that the character encoding of the NeXT computer can be mapped to whatever encoding is appropriate for the TEX fonts being used. This is supported through a file called `tex.remap` which is searched for in the current directory, the user's home directory, and then `/usr/lib/tex`.

Many array dimensions inside of TEX and METAFONT and various other programs have been increased; the hyphenation trie specifically has been made larger for support of multiple simultaneous foreign languages.

While this isn't a new feature, the use of `%&` *formatname* on the first line of your main TEX source file is an even better idea now that TEXview can invoke TEX directly. If you do not use this convention, TEX will not know what format to load.

2.2 Changes Shared Between TEXview and Dvips

Many features have been added to both TEXview and `dvips`; this is a list of the new features they share. Much more documentation is given in the chapter on `dvips`.

Included epsf images can be scaled such that their aspect ratio is distorted; simply give both a horizontal and vertical size (with `\epsfxsize` and `\epsfysize`) and the graphic will be scaled to fit.

Included PostScript graphics may be clipped to their bounding box by giving the command `\epsfclipon`; note that this may cause some graphics to get 'shaved' if they give a bounding box that is incorrect.

Included graphics file names that start with a back-tick (`) are interpreted instead as commands to run, and the actual data will be taken from stdout.

A new `MAKETEXPK` environment variable is supported.

There is now color support! Color in imported graphics was always supported, but now you can set the color for your TeX text, rules, and equations as well.

`MS-DOS`-style `.pfb` files are now supported; they are converted to ASCII in the output PostScript file. (Note that such files can not, of course, be distributed, since they contain copyrighted font code. This is not true if public domain fonts are used instead.)

The configuration file option '`p+`' has been added, meaning 'add these additional fonts to the resident font list'. This is in addition to the old '`p`' option, which gave a file name that completely replaced the resident font list.

The memory required by resident PostScript fonts has been dramatically reduced.

PostScript fonts can now be reencoded both at the PostScript and the virtual font level.

The desired paper size can be specified in the TEX source file; both `dvips` and TEXview will recognize these commands and attempt to comply with them.

2.3 Changes to Dvips

The following new features have been added to `dvips` since Release 2.0. These are the extensions that apply only to `dvips` and `afm2tfm`; more information can be found in the chapter on `dvips`.

Horizontal and vertical offsets can be specified in the configuration file.

Sequential page numbers can be specified by preceding the number with '='.

Paper size specials are now supported, as is paper size information in the configuration file that can be used to map requested paper sizes to available paper sizes. This is especially important for typesetters where wasted film can be expensive.

There is built-in command line help; just type `dvips` at a command line for a list of the options.

The ability to limit the number of pages in a section, and to put each section into a separate file, has been added. This makes it trivial to separate a 500 page book into sections of 20 pages for the Linotronic.

Crop marks can now be printed.

The desired page size can also be specified on the command line.

A new `-E` option will attempt to create an encapsulated PostScript file. This only works for documents in which you've selected one page. The calculated bounding box just takes into account rules and characters drawn on the page; no graphics are currently taken into account.

The `afm2tfm` program now prints out the entry in `psfonts.map` that should be added.

The `dvips` program now looks for a `%%VMUsage` comment and uses it; otherwise it approximates the amount of printer VM required by an included header or epsf file by the total size of the file.

A new `-b` option provides facilities for color separation, n-up printing, or poster printing.

Ligatures have now been turned off in all fixed-spaced fonts. Bad kern pairs in `AFM` files are now a warning rather than an error.

Accent ligatures in PostScript fonts have been removed.

2.4 Changes to TEXview

The following new features have been added to TEXview since Release 2.0. There is some sketchy documentation in the Hints panel; more documentation is in a later chapter of this manual.

There is now a Hints panel with some commonly-overlooked features of TeXview mentioned.

Graphics can be 'mocked'; if the hide graphics option is selected in Preferences, then gray boxes will be drawn instead of included graphics. This is for speed when the included graphics are very complex or use inefficient PostScript.

A console window has been added; this displays the results of any runs of `dvips` or METAFONT needed by TEXview. It automatically pops to front if a command takes longer than a few seconds, so the user no longer wonders why TEXview isn't responsive.

A 'tex' window has been added; this displays runs of TEX. Yes, now you can invoke TEX from within TEXview!

The arrow keys can now be 'qualified' with shift and alternate to reduce the amount of scroll. In addition, the left and right arrow keys move the document around on the page when they are so qualified.

Services support has been added. The initially available services are 'open this file' (takes a file name, returns nothing), 'reTEX the current file' (takes a file name but ignores it, just reTEX'ing the current file [this is because the current file might be a subfile—say, Chapter 2; better just to reTEX what you TEX'ed before] and returns nothing), 'reload `dvi` file' (takes a file name, again ignoring it, and returns nothing; this is useful if you run TEX from within `emacs`, for instance), and 'TEX eqn to EPS' that takes ASCII, runs TEX and

`dvips -E` on it and returns a PostScript clip. These services can be extended through the use of the `TeXview.service` file.

The TEXview window has a new, simpler look; the buttons have been moved to a separate command window. This way, those that want lots of big buttons can have them, and those that like a sparse, elegant interface can have that.

The page number field is now automatically enabled when a – or any digit is pressed. To go to a particular page, just make sure the main TEXview window is the key window and type in the page number followed by a return. (It's not enabled all the time because we want simple keystrokes to be interpreted as command keystrokes when possible to save some pinkies out there.)

FAX 'printing' should now work! (This was a tough one.)

You can now scroll the page around by click-dragging.

You can measure distances by clicking on the page; in the command window, the current click location and the distance from the last click location will be displayed. You can also change the units that these distances are displayed in.

The size, position, and exposure status (whether the window is visible or not) is now saved in the defaults database for the main window, the TEX window, the console window, and the command window. You must select 'Save Preferences' on the Preferences panel for this to take place.

TEXview now can open files with extensions of `.dvi` and `.tex`.

TEXview now responds to environment variables set in `~/.cshrc`. This is done by `system()`'ing a `csh` and reading the output from `printenv` and using that to augment any environment variables that live in the Workspace environment. When, oh when, will the Workspace get an environment?

No bitmapped fonts will be generated at resolutions below 69 dots per inch.

Many, many other features and changes to the user interface. Play and enjoy!

2.5 Incompatible Changes

The following incompatible changes have been made to NeXTTEX since Release 2.0.

You cannot mix and match old and new versions of TEXview and `dvips`. They use many of the same header files, and these header files have changed from 2.0 to 3.0.

PostScript fonts now use new, shorter (and perhaps less convenient) names; this was done in the interests of compatibility. See `/usr/lib/tex/inputs/dvips.tex` for more information.

TEXview no longer accepts file names on the command line; use the shell command `open` instead.

The `t` configuration file option (for paper size) is no longer supported. This is better done with the new paper size support.

2.6 Other Changes

The following additional changes have been made to NeXTTEX since Release 2.0. In addition to the changes given below, many, many bugs, big and small, have been squashed.

Landscape mode is now rotated 180 degrees to fit the `Preview` conventions. (Note that the landscape special, while still supported, should be done now with a papersize special.)

The PostScript accent fixes have been added to `psfonts.sty`; note that these may change depending on how the PostScript fonts are reencoded.

In path list expansion, any % substitutions are done on a per path element basis, rather than on the entire path list. Thus, before the path `.:/myfonts/%d/%n.%dpk` would always 'succeed' by opening the current directory as a `pk` file (and then it would blow up with an illegal command in the `pk` file); with the current release, this path list will work as you'd expect.

2.7 Known Problems

The following are known problems in NeXTTEX. They will be corrected in a future release.

You can't include epsf graphics that have an underscore in their name if you are also using a macro package that makes the underscore an active character. Few macro packages do this; when they do, simply rename your graphic file.

3. What is TEX?

TEX is a typesetting language and system developed by Professor Donald Knuth of Stanford University over the period 1977 to 1990. It was designed to be used in typesetting books and manuscripts, especially those containing much mathematics. The current version of the language has been stable since 1982, and is currently in use at thousands of computer installations all over the world.

The input to TEX is somewhat free format; it is emphatically not a WYSIWYG (what you see is what you get, often rephrased as what you see is all you get) system. The source can be created with any typical text editor, and for the most part spacing and line breaks in the input are ignored. Interspersed with the text are various typesetting commands that might, for instance, change fonts or skip to the next page. TEX is programmable with parameterized macros and user variables of a number of types, allowing the creation of macro packages that extend the power of TEX or make it more accessible.

TEX is not the solution to all the world's ills. As a document preparation system, it has three major deficiencies. First, it does not directly support graphics. You can use horizontal and vertical rules, and do some limited graphics using a variety of methods: LaTEX supports some graphics operations, and PostScript printer drivers allow inclusion of PostScript graphics. Secondly, TEX does not easily do complicated page layout, as might be required for newspapers, for instance. And finally, TEX does not allow interactive editing and viewing of the document as it is being developed.

All of these are design decisions; TEX was never intended to address any of these concerns. What TEX does, however, TEX does well. It supports a huge library of fonts, and additional fonts can be purchased from a number of companies. It supports full kerning and ligatures in its fonts. It has no equal when typesetting complicated mathematical formulas and displays. Its hyphenation algorithm is among the best in typesetting systems, and its line- and page-breaking algorithms are nothing short of incredible.

TEX is a complicated system, because typesetting is not a simple task. It is easy to generate quality documents without knowing much about TEX, but as your experience and requirements grow, you will find that TEX has the power, programmability and expandability to handle your most demanding needs.

One of the primary objectives of Knuth in writing TEX was computer and printer independence; the output you display on your NeXT screen should appear (within the resolution limits of the device) exactly the same as the output on the professional typesetting equipment of Addison-Wesley, for instance. The same source should work and generate identical output on all computer installations that run TEX.

To illustrate TEX, the processing of our sample `foo.tex` above is going to be examined, and each file used by TEX in the creation of the document will be explained. Another simple example of a TEX file is `story.tex`, from page 24 of *The TEXbook,*. Note that all source files for TEX should have the extension `.tex`.

Some of the commands used in `foo.tex` are not a part of primitive TEX; primitive TEX is very primitive indeed. Therefore, TEX by default loads a set of macros called 'plain TEX' into memory. Rather than load these macros in as TEX source (which it could), it instead loads them in a pre-digested form called a 'format file', with the extension `.fmt`. This `plain.fmt` file contains essentially a memory image of TEX after all of the macros in plain TEX are loaded.

Primitive TEX does not know any characteristics of any fonts, either. Therefore, a number of font metric files with the extension `.tfm` were loaded while the `plain.fmt` file was created. These files contain information about individual characters in the font and the font as a whole, such as the height, width, and depth of each character, what ligatures the font contains, and what kerning is necessary between which characters. There is one of these files for each font accessible to TEX, and they contain only information about the fonts that is independent of the output device.

As TEX was processing `foo.tex`, it created two files, one with the extension `.log` and one with the extension `.dvi`. The log file contains essentially a listing of the run as it was seen from the terminal, including any error messages that were displayed. This file is useful if an error message was too long and scrolled off the display, for instance; various switches can also be turned on inside TEX to display more information about what is happening during the typesetting process. The main output file of TEX is the device-independent file, with the extension `.dvi`. This file contains a description of the final document, including the location and font of each character used. The units used in this file are not based on any particular device, and the actual characters themselves are not described. It is up to another program, the 'driver', to interpret this `.dvi` file, look up the appropriate raster or outline representations of the individual characters, and create the final document.

The TEXview program is one such program. It 'drives' the NeXT screen, loading font raster information from a collection of packed, or `.pk`, files. There is one of these files for each font at each size, since the raster description of a character is by its very nature dependent on the resolution of the output device.

The METAFONT program is used to generate these packed font files at the appropriate resolutions from source descriptions. Normally, with NeXTTEX, you do not even need to be aware that METAFONT exists, except that sometimes TEXview or `dvips` will pause while they invoke METAFONT to generate a font. The Computer Modern fonts are the standard set of fonts used with TEX; the use of PostScript fonts is becoming more and more common.

This was intended to be a simple description of TEX. For more information, please refer to *The TEXbook* by Donald Knuth and *LaTEX User's Guide and Reference Manual* by Leslie Lamport, both published by Addison-Wesley Publishing Company and available in most college bookstores.

4. Acknowledgements

I would like to acknowledge Dr. Thomas Marchioro II, who showered me with suggestions, comments, support, coffee, code; without his help, the package would not be anywhere near as nice. If you can get this man to test your product, do so. Thanks, Tom!

Many other people were a great help. John Loyola did an excellent job of proofing this manual; any errors I'm sure I added after he finished. Susan Skulina, who will be my wife when you read this, put up with many lost weekends and evenings as I labored to add the 'last touch'. Janet Coursey went well over and above the call of duty in helping me with equipment problems, software problems, and just in general being the most incredible support person I have ever met; NeXT does not and cannot pay her enough for the job that she does. Dmitri Linde made some very useful suggestions and improvements, some of which I thought were extremely difficult; here is a man with a future. There are many, many other contributors who showered me with dozens of megabytes of mail messages, files that failed in one way or the other, suggestions, code, support, questions, and all sorts of other communications; please forgive me for not mentioning you all by name.

2 Using TEX

At this point you have TEX installed and have compiled some simple documents. It is time to get down and dirty with the details of TEX on the NeXT machine. In this chapter, we cover the special features of NeXTTEX that make it stand out above other implementations. We describe how to use different format files, including the supplied LaTEX, SliTEX, and your own local format files.

The distributed version of TEX is 3.141, which supports virtual fonts, multiple hyphenation tables, and a full eight-bit character set. In addition, the version of TEX is compiled with a very large memory size, allowing large macro packages, complex pages, and a large set of fonts to be used.

Information on using PostScript fonts and graphics is given in a later chapter on `dvips`.

Normally you do not interact directly with the TEX program itself; TEXview will call it for you and display the results. Nonetheless, it is often useful to invoke `tex` or `initex` yourself from a shell window; indeed, this is currently the only way to create a format file.

1. Exiting TEX

To exit the `tex` program at any prompt, simply hit control-D. TEX will respond with 'emergency exit'. If you are not at a prompt, hit control-C; this will interrupt TEX and should give you a prompt to which you can type control-D.

2. TEX Format Files

TEX for the NeXT machine is supplied as many different executables. Among these are `latex`, `slitex`, `virtex`, and `tex`. If these four files are examined closely, it will be noted that they are actually all the same file, but under four distinct names. The Unix operating system provides 'links' that allow files to have multiple names. The only difference between `tex` and `latex` is the format file they load.

The program TEX has only about 300 control sequences built in. These control sequences are very primitive and thus quite difficult to use. Therefore, before processing a

document, TEX loads in by default a macro package containing definitions that allow the document to be described at a higher level.

Parsing and processing the large amount of source in a typical macro package takes a long time. To eliminate this delay, the program `initex` can be used to create a 'format file' that contains essentially a memory image of TEX after all of the macros have been loaded. This format file loads relatively quickly, since it needs only to be copied into memory.

Thus, the executable for `tex` looks at the name under which it was invoked. It then loads a format file with the same name. Thus, invoking `tex` loads the `tex.fmt` file, which is linked to the `plain.fmt` file, which contains the default set of macros for plain TEX. Invoking `latex`, which is the exact same executable as `tex`, loads the `latex.fmt` file, which is linked to the `lplain.fmt` file, which contains the set of macros for LaTEX.

With this scheme it is easy to install a new format file and have it be loaded transparently. Let us say we have a local format file called `glib.fmt`, and we wish to create a command `glibtex` that will invoke a TEX that automatically loads `glib.fmt`. The following commands suffice.

```
localhost# cp glib.fmt /usr/lib/tex/macros
localhost# ln /usr/lib/tex/macros/glib.fmt /usr/lib/tex/macros/glibtex.fmt
localhost# ln /usr/bin/tex /usr/bin/glibtex
```

Of course, you have to be superuser to execute these commands.

2.1 Specifying a Format File on the Command Line

A format file can be specified on the command line. Simply give the name of the format file you wish to load after an ampersand (`&`) on the command line. For instance, we can automatically load `lplain.fmt` with the command

```
localhost> tex \&lplain
```

Note that with almost all command shells in Unix, the ampersand must be 'escaped' (preceded with a backslash) as in the above example.

2.2 Specifying a Format File in a TEX File

It is possible to specify the format file that should be loaded into TEX in your document file. This will override the format file name derived from the name of the executable under which TEX is invoked, and makes it easy to compile a given TEX document without trying to figure out first what format file it requires. To give this information, the first line of the TEX file should look like (for instance)

```
%&latex
```

which would indicate that the `latex.fmt` file should be loaded before processing the document. This specification must appear on the first line of the TEX source, and it must consist of the two characters `%&` followed immediately by the name of the format file.

This option is very important; without it, TEXview will not know what format file to use when it is told to 'recompile' or 'reTEX' the file.

3. Communicating with TeXview

To help reduce the edit–compile–view cycle with TEX, an option has been added to NeXTTEX that will automatically give TEXview the name of the current `dvi` file that TEX is processing. If you type

```
localhost> tex -v foo
```

and TEXview is running, then as soon as the first page of `foo` has been processed by TEX, it will automatically appear in the TEXview window. There is no need to wait until TEX is finished processing the file; as each page is completed by TEX, it becomes available for viewing from TEXview.

When TEX is automatically invoked from within TEXview, the `-v` option is supplied by default.

If the option is provided as `-V`, then TEXview will be launched if it is not already. (Launching TEXview can take several seconds, so be patient.)

The communication is effected with a named pipe called `.TeXview_Pipe` created in the user's home directory.

4. Invoking Unix Commands from TEX

In rare instances it might be desirable to execute a Unix command from TEX at a certain point in your document. For instance, this document sorts its index by executing such a command immediately before the index is typeset. NeXTTEX allows you to do this with output (write) stream number 18. For instance, the command (in TEX)

```
\immediate\write18{ls}
```

will cause the `ls` command to be executed at this point in the document. The output from the command will go to the current TEX window, and no output will actually be generated in the document. If you wish to include some output in your file, you can redirect the output to a file and `input` that file into your document, as in

```
\immediate\write18{date >datefile.tmp}
This document was typeset on \input datefile.tmp .
```

Note the space after the file name; this is to indicate to TEX the end of the file name. This particular example is much more easily accomplished with built-in TEX commands. Finally, the file name must have an extension (a portion of the file name following a period); if TEX sees a file name with no extension, it automatically appends a `.tex` to the filename.

Note that the `\immediate` prefix is necessary, otherwise TEX will delay the command until that particular page is flushed from TEX's memory, which is usually not what is desired.

Use of this extended feature of NeXTTEX will render your document less portable to other computer systems, since few have this particular extension. However, on other systems, the offending command will simply cause the command to be written to the terminal rather than executed.

5. IniTEX

The IniTEX program is used to create format files for use with TEX. To recreate the plain format file, for instance, simply type

```
localhost> initex plain \\dump
```

and after a lot of grinding you should have a brand new format file (named `plain.fmt`) in your current directory.

To load your own macro package on top of `plain`, all you need do is tell `initex` to load the regular `plain` format file before processing your macros:

```
localhost> initex \&plain mymacros \\dump
This is CTeX, NeXT Version 3.141 (INITEX)
(mymacros.tex)
Beginning to dump on file mymacros.fmt
.  .  .
```

Now you should copy this format file to the `/usr/lib/tex/formats` directory. It can be loaded by any of the methods listed in this chapter, such as by

```
localhost> tex \&mymacros bar
```

6. NeXT Environment Variables

There are several files TEX must look for when processing a document; these include format files, font metric files, and auxiliary input files. Since people might choose to move these files from their default places, or to use their own personal versions of the files, TEX reads several environment variables and uses them to search for these files. This section describes environment variables in general and lists the ones that are used by TEX.

The rest of this section assumes you are using the `csh` command shell or some other compatible shell. If you are using another shell, refer to documentation on that shell for how to set and display environment variables. Do not get these environment variables confused with shell variables; under `csh`, the latter are accessed by `set` and are usually lower case, while environment variables are almost always upper case.

An environment variable consists of two strings of characters: a name, and some contents. The `setenv` command can be used to change these environment variables. For instance, to set the environment variable `a` to `b`, you would type the command

```
localhost> setenv a b
```

To list the current environment variables, the command `setenv` without parameters works; type it now to verify that the above command worked. To delete an environment variable, use `unsetenv`.

Environment variables are local to each shell; setting an environment variable in one window does not affect the environment variables in another window. This can often lead to confusing, different results in different windows. For this and other reasons, environment variables are usually set in the shell initialization file; for `csh`, this file is named `.cshrc` in your home directory.

Most environment variables are paths. A path is a colon-separated list of directories to search for certain files. For instance, when TEX needs to load a format file, it searches the directory `.` (the current directory), followed by the directory `/usr/lib/tex/formats`, according to the paths listed below.

TEX examines the following environment variables. If they are not set, TEX uses the given default.

`TEXFORMATS` (`.:/usr/lib/tex/formats`) This is where TEX searches for format files.

`TEXINPUTS` (`.:/usr/lib/tex/inputs`) This is where TEX searches for input files. Input files also include any PostScript graphics that might be included in the document and any style files that may be used.

`TEXFONTS` (`.:/LocalLibrary/Fonts/TeXFonts/tfm:/usr/lib/tex/fonts/tfm`) This path is used when TEX is looking for `tfm` files.

`TEXPOOL` (`.:/usr/lib/tex`) This path is used by `initex` to find the `tex.pool` file.

`TEXEDIT` (`/usr/ucb/vi +%d %s`) This environment variable describes the command to execute if the `E` response is given to a TEX error prompt. All occurences of `%d` are replaced by the line number at which the error occurred, and all occurrences of `%s` are replaced by the name of the file in which the error occurred. If you prefer `Edit` for your editor, set this command to '`/usr/bin/openfile %s:%d`'.

7. The New Font Selection Scheme

The new font selection scheme, also called `NFSS`, will work fine with NeXTTEX. It is not distributed because it has not yet stabilized but such support is planned for the future.

8. International Character Sets

In addition to the normal 95-character ASCII character set, the NeXT computer supports about 124 additional characters accessible from the keyboard. Most of these characters are for international language support. The most important of these characters are the accented letters and a few additional characters such as Æ and Ø.

TEX also supports many of these characters. There are two ways TEX can support these characters: through special macros and accents (as is most commonly done, and must be done with versions of TEX prior to 3.0) and with the new 3.0 eight-bit support and remapping. We shall address each of these in turn.

Note that many editors and standard Unix programs do not support eight-bit characters; the extended character set can give fits to standard Unix shells and tools. Experiment with various editors and tools to determine which work and which don't, and then use the ones that work if you need to use the extended characters.

8.1 Poor Man's Extended Characters

The easiest way to get the NeXT extended characters to work is to include the file `next.tex` at the top of your file with a line like

```
\input next
```

(for plain TEX) or with the addition of `next` as a document style option in LaTEX:

```
\documentstyle[next]{article}
```

The included file can be moved along with the TEX documents for maximum portability. What this file does is make many of the upper eight-bit characters 'active', meaning that

they directly invoke a macro that typesets what is requested. For instance, the NeXT character set has at character position 214 the character á. This can be typed on the standard English keyboard in most editors with alternate-'e' followed by 'a'; some editors do not support these characters. To typeset this character with TEX, we want to use the sequence `\'a`. So `next.tex` (and `next.sty`) just make character position 214 active and equal to a macro whose body is `\'a`.

Most of the extended characters on the NeXT are supported this way; these include (the unbreakable space) À Á Â Ã Ä Å Ç È É Ê Ë Ì Í Î Ï Ñ Ò Ó Ô Õ Ö Ù Ú Û Ü Ý μ × ÷ © ¡ £ / ƒ § ’ “ fi fl – † ‡ · | ¶ • , ,, ” ... ¬ ¿ ¹ ` ´ ˆ ˜ ¯ ˘ ˙ ¨ ² ˚ ¸ ³ ˝ ˇ — ± ¼ ½ ¾ à á â ã ä å ç è é ê ë ì Æ í ª î ï ñ Ł Ø Œ º ò ó ô õ ö æ ù ú û ı ü ý ł ø œ ß ÿ

The characters that are not supported in this fashion are capital eth, capital thorn, cents, yen, currency, registered, all guillemot characters, per thousand, ogonek, thorn, and eth. This is because TEX has no standard support for these characters in its fonts.

The advantage of this approach is that it works with the existing software and fonts, both PostScript and Computer Modern (for the most part.) The disadvantages, however, is that these characters are not treated as just simple characters; this is true of all accent constructions in TEX, anyway. Hyphenation of works containing these extended characters will not work, and these extended characters cannot be used in file names, font names, or macro names.

8.2 Extended Characters Done the Right Way

The TEX community is hard at work at fixing this deficiency. The computer software is all in place to support it; various people and groups are working on replacements for the Computer Modern bitmapped fonts and new macro packages, hyphenation patterns, and other necessary parts to fully support these extended characters.

The NeXTTEX software has been designed to support these standards when they become available. The primary problem is the different eight-bit character sets adopted by various parts. The NeXT computer has one character set for the keyboard. TEX internally uses another. Various bitmapped fonts use a third, and PostScript fonts can use any of a number of character sets. Somehow the characters the user types need to be translated to the corresponding character internally for TEX, and then that TEX character code needs to be converted to something appropriate for the font in use.

This is all explained in more detail in a later chapter. For now, we shall limit ourselves to discussing how the characters that the user types can be remapped to different character codes internally for TEX.

This support is provided through a file called `tex.remap`, which is searched for in the current directory, the user's home directory, and then `/usr/lib/tex`. This file is a list of lines, each of the following form:

ext: *int int int* ...

where *ext* is the external character code (1 through 255) and the *int*s represent the internal code for that (and the successive) external characters. Thus, to remap character 128 (externally) to 200 and 129 to 211, a line such as

```
128:  200 211
```

would do. If you use such remapping, be sure to run `initex` to create new format files; this will allow TEX to display as well as use these characters.

We have provided an example mapping file in `/usr/lib/tex/src/misc/tex.remap` that maps from the standard NeXT encoding to the Extended TEX encoding that is expected to become the standard for TEX. More details on remapping are given later in the manual.

3 Printing TEX Documents

This chapter is documentation on the `dvips` driver program that generates PostScript from TEX documents. Note that this documentation is general, shared among all implementations of TEX that use `dvips`, and thus some of it may seem inappropriate. On almost all other systems, `dvips` is invoked via a command shell; on the NeXT, `dvips` is most often invoked automatically by the Print menu item in TEXview. This chapter is relevant, however, for details on graphics inclusion, on font naming, and on the extra options you can specify for `dvips` in the TEXview print panel.

Note that many the `dvips` options that you want on by default can be specified in a file named `.dvipsrc` in your home directory.

The `dvips` program converts a TEX `dvi` file into a PostScript file for printing or distribution. Seldom has such a seemingly easy programming task required so much effort.

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. Installation is covered in section 14 near the end of this document.

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 a different resolution 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 æsthetics 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 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. The `dvips` driver 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. Using dvips

To use `dvips`, simply type

```
localhost> 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; they are described in a later section. For a brief summary of available options, just type

```
localhost> dvips
```

3. Paper Size and Landscape Mode

Most TEX documents at a particular site are designed to use the standard paper size (for example, letter size in the United States or 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.

`\ifdim#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.

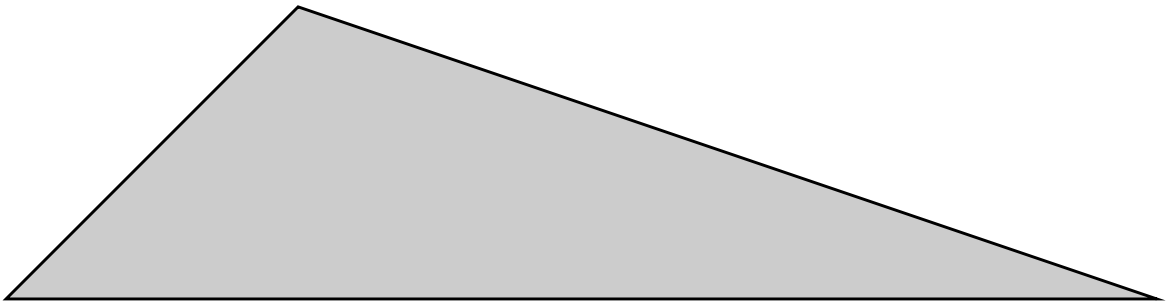
For these and all other header files (including the headers required by `dvips` itself and any downloaded fonts), the printer VM budget is debited by some value. If the header file has, in its first 1024 bytes, a line of the form

`%%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 command that starts with a double quote (`"`). 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}}
```

(Note that 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 supported by `dvips`.

Other facilities are available for historical reasons, but their use is discouraged, in hope that some 'sane' form of PostScript inclusion shall become standard. Note that 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 tortuous 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 `dvi2ps` programs. Yet.

5. Using PostScript Fonts

Thanks to Donald E. Knuth, the `dvips` driver now supports PostScript fonts through the virtual font capability. PostScript fonts are (or should be) accompanied by a font metric file such as `Times-Roman.afm`, which describes characteristics of a font called Times-Roman. To use such fonts with TEX, we need `tfm` files that contain similar information. These can be generated from `afm` files by running the program `afm2tfm`, supplied with `dvips`. This program also creates virtual fonts with which you can use normal plain TEX conventions.

Note that non-resident downloaded PostScript fonts tend to use a great deal of printer virtual memory. PostScript printers typically have between 130,000 and 400,000 bytes of available virtual memory; each downloaded font will require approximately 30,000 bytes of that. For many applications, bitmapped fonts work much better, even at typesetter resolutions (with the `-Z` option.)

Even resident PostScript fonts can take a fair amount of memory, but less with this release of `dvips` than previously. Also, bitmapped fonts tend to image faster than PostScript fonts.

5.1 The afm2tfm Program

The `afm2tfm` program can convert an Adobe `afm` file into a 'raw' TEX `tfm` file with the command

```
localhost> afm2tfm Times-Roman rptmr
```

(You should run this from in a directory where `Times-Roman.afm` resides.) The output file `rptmr.tfm` is 'raw' because it does no character remapping; it simply converts the character information on a one-to-one basis to TEX characters *with the same code*. The name `rptmr` stands for Resident PostScript Times Roman; section 6 below explains more about a proposed scheme for font names.

In the following examples, we will use the font Times-Roman to illustrate the conversion process. For the standard 35 LaserWriter fonts, however, it is highly recommended that you use the supplied `tfm` and `vf` files that come with `dvips` (usually in a file called `dvipslib.tar.Z`), as these files contain some additional changes that make them work better with TEX than they otherwise would.

PostScript fonts have a different encoding scheme from that of plain TEX. Although both schemes are based on ASCII, special characters such as ligatures and accents are handled quite differently. Therefore we obtain best results by using a 'virtual font' interface, which makes TEX act as if the PostScript font had a standard TEX encoding. Such a virtual font can be obtained, for example, by the command

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

This produces two outputs, namely the 'virtual property list' file `ptmr.vpl`, and the TEX font metric file `rptmr.tfm`. The latter file describes an 'actual font' on which the virtual font is based.

To use the font in TEX, you should first run

```
localhost> vptovf ptmr.vpl ptmr.vf ptmr.tfm
```

and then install the virtual font file `ptmr.vf` in the virtual font directory (by default, `/usr/lib/tex/fonts/vf`) and install `ptmr.tfm` and `rptmr.tfm` in the directory for TEX font metrics (by default, `/usr/lib/tex/fonts/tfm`). (This probably has already been done for you for the most common PostScript fonts.) 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. Once this has been done, you're all set. You can use code like this in TEX henceforth:

```
\font\myfont=ptmr at 10pt
\myfont Hello, I am being typeset in Times-Roman.
```

Note that there are two fonts, one actual ('rptmr', which is analogous to a raw piece of hardware) and one virtual ('ptmr', which has typesetting know-how added). You could also say

```
\font\TR=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 called `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). Experts may refer to Donald E. Knuth article in *TUGboat* v. 11, no. 1, Apr. 1990, pp. 13–23. "Virtual Fonts: More Fun for Grand Wizards."

When `dvips` goes to use a font, it first checks to see if it is one of the fonts listed in a file called `psfonts.map`. If it is mentioned in that file, then it is assumed that the font is a resident PostScript font and can be found with the PostScript `findfont` operator. This file resides by default in `/usr/lib/tex/ps`, and consists of a single font per line. Note that only the raw PostScript font names should be listed in this file; the `vf` fonts should not be, since they are automatically mapped to the raw PostScript fonts by the virtual font machinery. The supplied `psfonts.map` file defines the most common PostScript fonts.

As much as possible, the PostScript fonts follow plain TEX conventions for accents. The two exceptions to this are the Hungarian umlaut (which is at position `0x7D` in `cmr10`, but position `0xCD` in `ptmr`) and the dot accent (at positions `0x5F` and `0xC7`, respectively). 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. Note that these definitions will not work with normal TEX fonts for the relevant accents; note also that these definitions are already part of the distributed `psfonts.sty`. In addition, the `\AA` that is used to typeset the Å character must be modified as shown.

```
\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}
```

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

5.2 Changing a Font's Encoding

The `afm2tfm` program also allows you to specify a different encoding for a PostScript font. This should only be done by wizards. This can be done at two levels.

You can specify a different output encoding with `-o`. This only applies when you are building a virtual font, and it tells `afm2tfm` to attempt to remap the font to match the output encoding as closely as possible. In such an output encoding, you can also specify ligature pairs and kerning information that will be used in addition to the information in the `afm` file. This will be the most common re-encoding required, since the only thing that changes is the `vf` file (and its associated `tfm` file) and since most everything you would want to do can be done with this method.

You can also specify a different PostScript encoding with `-p`. This option affects the generation of both the raw `tfm` file and the virtual `vf` and `tfm` files, and it also requires that the encoding file be available to be downloaded as part of every PostScript document produced that uses this font. But this is the only way to access characters in a PostScript font that are neither encoded in the `afm` file nor built from other characters (constructed characters.) For instance, `Times-Roman` contains the extra characters `registered` and `thorn` (among others) that can only be accessed through such a PostScript reencoding. Any ligature or kern information specified in the PostScript encoding is ignored by `afm2tfm`.

The format of the encoding files is simple—it is precisely the same format that is used to define an encoding vector in PostScript! For this reason, the same file can be used as a PostScript or TEX encoding file for `afm2tfm` as well as downloaded to the printer as part of a document that uses a reencoded font.

The specific format that `afm2tfm` accepts is one of the following form:

```
% comments are mostly ignored; more on this later
/MyEncoding [ /Alpha /Beta /Gamma /Delta ...
     /A /B ... /Z % exactly 256 entries, each with a / at the front
     /wfooaccent /xfooaccent /yfooaccent /zfooaccent ] def
```

Comments, which start with a percent sign and continue until the end of the current line, are mostly ignored. The first 'word' of the file must start with a forward slash (a PostScript literal name) and is used as the name of the encoding. The next word must be an open bracket. Following that must be precisely 256 character names; use `/.notdef` for any that you do not want to define. Finally, there must be a close bracket. The final token is usually `def`, but this is not enforced. Note that all names are case sensitive.

Any ligature or kern information is given in the comments. As each comment is encountered, it is examined. If the first word after the percent sign is `LIGKERN`, exactly, 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 on the end of a line.)

In these `LIGKERN` statements, three types of information may be specified. These three types are ligature pairs, kerns to remove or ignore, and the character value of this font's boundary character. Which of the types the particular set of words corresponds to is automatically determined by the allowable syntax.

Throughout the `LIGKERN` section, the boundary character is specified as `||`. To set the boundary character value, a command such as `|| = 39 ;` must be used.

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. Normally 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 space l =: lslash ; space L =: Lslash ;
% 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 ; zero {} * ; * {} zero ;
% LIGKERN one {} * ; * {} one ; two {} * ; * {} two ;
% LIGKERN three {} * ; * {} three ; four {} * ; * {} four ;
% LIGKERN five {} * ; * {} five ; six {} * ; * {} six ;
% LIGKERN seven {} * ; * {} seven ; eight {} * ; * {} eight ;
% LIGKERN nine {} * ; * {} nine ;
```

5.3 Special Effects

Special effects are also obtainable, with commands such as

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

which create `ptmro.vpl` and `rptmro.tfm`. To use this, proceed as above but put the line

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

into `psfonts.map`. Then `rptmro` (our name for an obliqued Times) will act as if it were a resident font, although it is actually constructed from Times-Roman by PostScript hackery. (This oblique version of Times-Roman is obtained by slanting everything 1/6 to the right.) Similarly, you can get an expanded font by

```
localhost> 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

```
localhost> afm2tfm Times-Roman -V ptmrc rptmr
```

This is done strictly with a virtual font, however. In addition, the font on which the small caps font is based (in this case `rptmr` may already be created and installed, in which case no additional `psfonts.map` entry is needed. In any case, you must give the appropriate name of the font that is not small caps as the base name (last parameter) to `afm2tfm`. For instance, if you create a slanted small caps font, you must give the base name of the raw slanted font as that last parameter, not the base name of the unslanted font.

By default, the `-V` option uses a font scaled to 80% for lower case. If you specify the `-c` option, you can change this scaling.

If you change the PostScript encoding of a font, you must specify the input file as a header file, as well as give a reencoding command. For instance, let us say we are using Times-Roman reencoded according to the encoding `MyEncoding` (stored in the file `myenc.enc`) as `rptmrx`. In this case, our `psfonts.map` entry would look like

```
rptmrx Times-Roman "MyEncoding ReEncodeFont" <myenc.enc
```

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. Note that 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.

The command line switches to `afm2tfm` are:

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

`-c` *scale* If this option is given when creating a small caps font (with `-V`), then the scaling for the 'lower' case will be changed from the default 0.8 to the fraction given here.

`-O` This option forces all character designations in the resultant `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.

`-p` *file* This specifies a file to use for the PostScript encoding of the font. Note that this file must also be mentioned as a header file for the font in `psfonts.map`, and that ligature and kern information in this file is ignored.

`-s` *slant* All characters are slanted to the right by the stated slant; if it 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 this file; the file is not needed once the `vf` file has been created.

`-T` *file* This option specifies that *file* is to be used for both the PostScript and target TEX encodings of the font.

`-v` *file* Generate a virtual property list `vpl` 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.

5.4 Non-Resident PostScript Fonts

If you want to use a non-printer-resident PostScript font for which you have a `pfb` or `pfa` file (an Adobe Type 1 font program), you can make it act like a resident font by putting a '`<`' sign and the name of the `pfb` or `pfa` file just after the font name in the `psfonts.map` file entry. For example,

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

will cause `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. Similarly, you can generate transformed fonts and include lines like

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

in `psfonts.map`, in which case `StoneInformal.pfb` will be loaded whenever StoneInformal-Condensed is used. (Each header file is loaded at most once per PostScript file. The `pfb` files should be installed in the `dvips` header directory [usually `/usr/lib/tex/ps`] with the other header files.)

If you are using a `pfb` file that has different PostScript encodings, you would need to multiple header files for that font in `psfonts.map`. If, for instance, `StoneInformal` was both non-resident and you wanted to reencode it in PostScript with `MyEncoding` stored in `myenc.enc`, a line such as

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

When using such files, `dvips` is smart enough to unpack the standard binary `pfb` format into ASCII so there is no need to perform this conversion yourself. In addition, it will scan the font to determine its memory usage, as it would for any header file.

5.5 Font Aliases

Some systems don't handle files with long names well—MS-DOS is a notable example. For this reason, `dvips` will accept an alias for such fonts. Such an alias should be the first word in the `psfonts.map` line. For instance, if we wanted the name `rptmr` to be used for the raw `Times-Roman` since our computer can't handle long names or, alternatively, we want to follow the standard naming conventions, we would use the following line in our `psfonts.map` file:

```
rptmr Times-Roman
```

The `tfm` file must have the name `rptmr.tfm`.

The distribution file `adobe` contains a list of the short names that should be used for most Adobe fonts currently available. Please reference this file when installing a new font and use the standard name.

The parsing of the `psfonts.map` file should be explained to eliminate all confusion. If a line is empty or begins with a space, asterisk, semicolon, or hash mark, it is ignored. Each remaining line is separated into words, where words are separated by spaces or tabs. 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 always the name TEX uses for the font and is also the name of the raw `tfm` file. If there is another name word, that name is used as the PostScript name; if there is only one name word, it is used for both the TEX name and the PostScript name.

Note that `dvips` no longer registers the full PostScript name if an alias is given, so the single line

```
rptmr Times-Roman
```

would only allow `dvips` to find the `rptmr` font and not the `Times-Roman` font.

6. Font Naming Conventions

This section of the manual has been written by Karl Berry and specifies a standard for naming fonts for TEX. This standard has been adopted in `dvips`, and it is recommended that it be followed where possible.

As more typeface families become available for use with TEX, the need for a consistent, rational naming scheme for the font filenames concomitantly grows. Some (electronic) discussion has gone into the following proposal; I felt it was appropriate now to bring it before a wider community. In some respects, it follows and simplifies Mittelbach and Schöpf's article in *TUGboat*, volume 11, number 2 (June 1990).

Here are some facts about fonts that went into the hopper when creating this proposal:

- TEX runs on virtually all computers, under almost as many operating systems, all with their own idea of how files should be named. Any proposal regarding filenames, therefore, must cater to the lowest common denominator. That seems to be eight characters in length, not counting any extension, and with case being insignificant. Characters other than letters and numerals are probably unusable.

- Most typefaces are offered by several vendors. The version offered by vendor A is not compatible with that of vendor B.

- Typefaces typically come in different weights (hairline to extra heavy), different expansions (ultra condensed to wide), and an open-ended range of variants (italic, sans serif, typewriter, shadow, ...). No accepted standards exist for any of these qualities, nor are any standards ever likely to gain acceptance.

- The Computer Modern typeface family preserves traditional typesetting practice in at least one important respect: different sizes of the same font are not scaled linearly. This is in contrast to most commercial fonts available.

Here is how I propose to divide up the eight characters:

```
FTTWVEDD
```

where

- `F` represents the foundry that produced the font, and is omitted if there isn't one.
- `TT` represents the typeface name.
- `W` represents the weight.
- `V` represents the variant, and is omitted if both it and the expansion are "normal".
- `E` represents the expansion, and is omitted if it is "normal".
- `DD` represents the design size, and is omitted if the font is linearly scaled from a single `tfm` file.

  See the section on virtual fonts (towards the end) for an exception to the above.

The weight, variant, and expansion are probably all best taken from the original source of the typeface, instead of trying to relate them to some external standard.

Before giving the lists of abbreviations, let me point out two problems, to neither of which I have a good solution. 1) Assuming that only the English letters are used, two letters is enough for only 676 typeface families (even assuming we want to use all possible combinations, which is doubtful). There are many more than 676 typeface families in the world. 2) Fonts with design sizes over 100 pt are not common, but neither are they unheard of.

On to the specifics of the lists. If you adopt this proposal at your own installation, and find that you have fonts with some property I missed, please write to me (see the end of the article for various addresses), so I can update the lists. You can get the most up-to-date version of these lists electronically, by anonymous ftp from the host `ftp.cs.umb.edu`. I will also send them to you by electronic mail, if necessary.

I give the letters in lowercase, which is preferred on systems where case is significant. Most lists are in alphabetical order by the abbreviations.

6.1 Foundry

This is the current list of foundries.

`a` Autologic
`b` Bitstream
`c` Agfa-Compugraphic
`g` Free Software Foundation (`g` for GNU)
`h` Bigelow & Holmes (with apologies to Chuck)
`i` International Typeface Corporation
`p` Adobe (`p` for PostScript)
`r` reserved for use with virtual fonts; see below
`s` Sun

6.2 Typeface Families

The list of typefaces is:

`ad` Adobe Garamond
`ag` Avant Garde
`ao` Antique Olive
`at` American Typewriter
`bb` Bembo
`bd` Bodoni
`bg` Benguiat
`bk` Bookman
`bl` Balloon
`bv` Baskerville
`bw` Broadway
`cb` Cooper Black
`cl` Cloister
`cr` Courier
`cn` Century
`cs` Century Schoolbook
`hv` Helvetica
`gm` Garamond
`go` Goudy Oldstyle
`gs` Gill Sans
`jo` Joanna
`lc` Lucida
`lt` Lutetia
`nc` New Century Schoolbook
`op` Optima
`pl` Palatino
`pp` Perpetua
`rw` Rockwell
`st` Stone
`sy` Symbol
`tm` Times
`un` Univers
`uy` University
`zc` Zapf Chancery
`zd` Zapf Dingbats

6.3 Weight

This is a list of the possible weights, roughly in order from lightest to heaviest.

`a`	hairline	`d`	demi
`t`	thin	`s`	semi
`i`	extra light	`b`	bold
`l`	light	`x`	extra bold
`k`	book	`h`	heavy
`r`	regular	`c`	black
`m`	medium	`u`	ultra

6.4 Variants

The variants are:

`a`	alternate	`n`	informal
`b`	bright	`o`	oblique (i.e., slanted)
`c`	small caps	`r`	normal (roman or sans)
`e`	engraved	`s`	sans serif
`g`	grooved (as in the IBM logo)	`t`	typewriter
`h`	shadow	`u`	unslanted italic
`i`	(text) italic	`x`	expert
`l`	outline		

If the variant is `r`, and the expansion is also normal, both the variant and the expansion are omitted. When the normal version of the typeface is sans serif (e.g., Helvetica), `r` should be used, not `s`. Use `s` only when the typeface family has both serif and sans serif variants. The "alternate" variant (`a`) is used by some Adobe fonts that have spiffy swashes and additional ligatures. The "expert" variant (`x`) is also used by some Adobe fonts with oldstyle figures and small caps.

Some fonts have multiple variants; Stone Informal Italic, for example. The only reasonable approach to these is to list all the letters for all the variants, choosing one to end with that is not also an expansion letter. Of course, it is possible that the name will become too long if you do this, but ... well, I'm open to suggestions. It's also possible the name will be ambiguous, if some new letter is used for expansions in the future. You can avoid this problem by adding the expansion `r` (if it doesn't make the name too long), and never using `r` for the last variant.

6.5 Expansion

This is a list of the possible expansions, in order from narrowest to widest.

`o`	extra condensed	`x`	extended (by hand)
`c`	condensed (by hand)	`e`	expanded (automatic)
`n`	narrow (automatic)	`w`	wide
`r`	regular, normal, medium (usually omitted)		

Expansion of fonts is sometimes done automatically (as in PostScript **scale**), and sometimes done by humans. I chose 'narrow' and 'expanded' to imply the former, and 'condensed' and 'extended' to imply the latter, as I believe this reflects common usage.

6.6 Naming Virtual Fonts

In concert with releasing TEX 3.0 and METAFONT 2.0, Don Knuth wrote two new utility programs: `VFtoVP` and `VPtoVF`, which convert to and from "virtual" fonts. Virtual fonts provide a general interface between the writers of TEX macros and font suppliers. In general, therefore, it is impossible to come up with a general scheme for naming virtual fonts, since each virtual font is an individual creation, possibly bringing together many unrelated fonts.

Nevertheless, one common case is to use virtual fonts to map TEX's default accent and other character code conventions onto a vendor-supplied font. For example, `dvips` does this for fonts given in the PostScript "standard encoding". In this case, each font consists of a "virtual" `tfm` file, which is what TEX uses, a "raw" `tfm` file, which corresponds to the actual device font, and a `vf` file, which describes the relationship between the two.

This adds another dimension to the space of font names, namely, "virtualness" (or rather, "rawness", since it is the virtual `tfm` files that the users want to see). But we have already used up all eight characters in the font names.

The best solution I have been able to think of is this: prepend `r` to the raw `tfm` files; the virtual `tfm` files should be named with the usual foundry prefix. For example, the virtual Times-Roman tfm file is named `ptmr`, as usual; the raw Times-Roman `tfm` file is named `rptmr`. To prevent intolerable confusion, I promise never to give a foundry the letter `r`.

This scheme will work only as long as the virtualized fonts do not have design sizes; if they do, another foundry letter will have to be allocated, it seems to me.

A pox upon the houses of those who decided on fixed-length filenames!

6.7 Examples

In closing, I will give two examples. First, the fonts in the Univers typeface family were assigned numbers by its designer, Adrien Frutiger. (You can see the scheme on, for example, page 29 of *The Art of Typo.icon.ography*, by Martin Solomon.) Naturally, we want to give them names.

`unl` 45 (light)
`unli` 46 (light italic)
`unlrc` 47 (light condensed)
`unlic` 48 (light condensed italic)
`unlro` 49 (light extra condensed)
`unmrx` 53 (medium extended)
`unm` 55 (medium)
`unmi` 56 (medium italic)
`unmrc` 57 (medium condensed)
`unmic` 58 (medium condensed italic)
`unmro` 59 (medium extra condensed)
`undrx` 63 (demibold extended)
`und` 65 (demibold)
`undi` 66 (demibold italic)
`undrc` 67 (demibold condensed)
`undic` 68 (demibold condensed italic)
`unbrx` 73 (bold extended)
`unb` 75 (bold)
`unbi` 76 (bold italic)
`unxrx` 83 (extra bold extended)

Second, here are names for the standard PostScript fonts and their variants: Fonts marked by an asterisk do not require using virtual fonts; the raw fonts can be used directly because no remapping is necessary; every character is encoded.

`pagk` AvantGarde-Book
`pagkc` AvantGarde-Book (Small Caps)
`pagko` AvantGarde-BookOblique
`pagd` AvantGarde-Demi
`pagdo` AvantGarde-DemiOblique
`pbkd` Bookman-Demi
`pbkdi` Bookman-DemiItalic
`pbkl` Bookman-Light
`pbkli` Bookman-LightItalic
`pbklc` Bookman-Light (Small Caps)
`pcrb` Courier-Bold
`pcrbo` Courier-BoldOblique
`pcrro` Courier-Oblique
`pcrr` Courier
`phvb` Helvetica-Bold
`phvbo` Helvetica-BoldOblique
`phvro` Helvetica-Oblique
`phvr` Helvetica
`phvrc` Helvetica (Small Caps)
`phvbrn` Helvetica-Narrow-Bold
`phvbon` Helvetica-Narrow-BoldOblique
`phvron` Helvetica-Narrow-Oblique
`phvrrn` Helvetica-Narrow
`pncb` NewCenturySchlbk-Bold
`pncbi` NewCenturySchlbk-BoldItalic
`pncri` NewCenturySchlbk-Italic
`pncr` NewCenturySchlbk
`pncrc` NewCenturySchlbk (Small Caps)
`pplb` Palatino-Bold
`pplbi` Palatino-BoldItalic
`pplbu` Palatino-BoldUnslanted
`pplrrn` Palatino-Narrow
`pplrre` Palatino-Expanded
`pplri` Palatino-Italic
`pplr` Palatino
`pplro` Palatino-Oblique
`pplru` Palatino-Unslanted
`pplrc` Palatino (Small Caps)
`psyr` Symbol*
`psyro` Symbol-Oblique*
`ptmb` Times-Bold
`ptmbi` Times-BoldItalic
`ptmrrn` Times-Narrow
`ptmrre` Times-Expanded
`ptmri` Times-Italic
`ptmro` Times-Oblique
`ptmr` Times-Roman
`ptmrc` Times-Roman (Small Caps)
`pzcmi` ZapfChancery-MediumItalic
`pzdr` ZapfDingbats*

Please contact Karl Berry if you have any comments or additions. Karl can be reached at `karl@cs.umb.edu`, or at 135 Center Hill Road, Plymouth, MA 02360.

7. Command Line Options

The `dvips` driver 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.

```
    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 X # Horizontal resolution
 t s Paper format                   Y # Vertical resolution
 x # Override dvi magnification     Z*  Compress bitmap fonts
    # = number   f = file   s = string  * = suffix, '0' to turn off
    c = comma-separated dimension pair (e.g., 3.2in,-32.1cm)
```

`-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.

`-c` *num*: Generate *num* copies of every page, by using PostScript's `#copies` feature. Default is 1. (For collated copies, see the `-C` option below.)

`-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. The source file `debug.h` indicates what the values of *num* can be, or see section 15 of this manual. Use a value of −1 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, and 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 this option if you want both.

`-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.

`-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 an exclamation mark, 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. This option also disables the automatic reading of the `PRINTER` environment variable, and 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 this option if you want both.

`-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. Another form of page selection is available by using `-pp` followed by a comma-separated list of pages or page-ranges, where the page ranges are colon-separated pairs of numbers. Thus, you can print pages 3–10, 21, and 73–92 with the option `-pp 3:10,21,73:92`.

`-q`: Run in quiet mode. 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. (Note, of course, that `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 on by default because PostScript previewers and spoolers still have problems with the structuring conventions.

`-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 and deleted 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.

`-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.

`-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.

8. Configuration File Searching

The `dvips` program has a system of loading configuration files such that certain parameters can be set globally across the system, others can be set on a per-printer basis, and yet others can be set by the 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 by default `.:/usr/lib/tex/ps`. 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 anything seen 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.

Note that because the printer-specific configuration files are read after the user's configuration file, the user's `.dvipsrc` 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 personalized printer-specific configuration files that will be found before the system global ones.

If your printer uses a different resolution than 300 dpi, make sure that you have given a METAFONT mode as well as a resolution in the printer configuration file. Also make sure that METAFONT knows about the mode, by entering it into your local `mode_def` file (typically `waits.mf`; `amiga.mf` on the Amiga, `next.mf` on the NeXT) and recreating the `plain.base` file for METAFONT, including the `mode_def` file. (Another good mode definition file is `modes.mf` by Karl Berry, which is available from `ftp.cs.umb.edu` in `pub/tex/modes.mf`.) The most common problem in generating fonts with METAFONT is that this file with the mode definitions is not included when creating the `plain.base` file.

9. Configuration File Options

Most of the configuration file options are similar to the command line options, but there are a few new ones.

Again, many may be turned off by suffixing the letter with a zero (0). These options are `a`, `f`, `q`, `r`, `I`, `K`, `N`, `U`, and `Z`.

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 PostScript's `#copies`. 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 in quiet mode 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*: Execute the system command listed, for example as a UNIX shell command. Execution takes place immediately, while the configuration file is being read. This option can be used to insert the current date into a header file, for instance, as explained at the end of section 13. Possibly dangerous; in many installations it 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*.

`I`: Ignore the `PRINTER` environment variable.

`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. This is passed along to `MakeTeXPK` and overrides mode derivation from the base resolution.

`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.

`P` *path*: The (colon-separated) path to search for bitmap `pk` font files is *path*. The `TEXPKS` environment variable will override this. If a `%` character is found in *path*, the following substitutions will be made, and then a search will be made for the resulting filenames. A `%f` is replaced by the font name. A `%b` is replaced by the output device horizontal resolution dots per inch. A `%d` is replaced by the font size in dots per inch. A `%p` is replaced by the font family; this is always `pk`. A `%m` is replaced by the font mode; this

is the mode given in the `M` option. Note that, for each path element, if it contains a percent sign, you must give the full file name, including path, rather than just the directory name; a path element such as `/fonts/%b` will try to open `/fonts/300` when looking for `cmr10.329pk`, for instance, and this may not be what is intended; `/fonts/%b/%f.%dpk` is needed. If a path element does not contain a percent sign, there is no need to specify the entire file name (because you can't, unless you list all possible specific font names!).

`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. Note that the resultant output will 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 `TEXINPUTS` environment variable will override this.

`T` *path*: The path to search for the `tfm` files is *path*. The `TEXFONTS` environment variable will override this. This path is used for resident fonts and fonts that can't otherwise be found. It's usually best to make it identical to the path used by TEX.

`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*: Sends *string* to stderr, if a parameter is given; otherwise it cancels another previous 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 that the resultant output has special characteristics.

`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.

10. 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. The `dvips` program 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. The `MakeTeXPK` script supplied invokes METAFONT 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. Note that the cache directory must exist on the `pk` file search path in order for `MakeTeXPK` to work.

11. Path Interpretation

The `dvips` program needs to read a wide variety of files from a large set of directories. It uses a set of paths to do this. The actual paths are listed in the next section; this section describes how the paths are interpreted.

All path variables are names of directories (path elements), separated by colons. Each path element can be either the literal name of a directory or one of the `~` forms common under UNIX. If a path element is a single tilde, it is replaced by the contents of the environment variable `HOME`, which is normally set to the user's home directory. If the path element is a tilde followed by anything, the part after the tilde is interpreted as a user name, and his home directory is fetched from the system password file and used as the real path element.

Where environment variables can override paths, an additional path element form is allowed. If a path element is the empty string, it is replaced with the system defaults. Thus, to search the user's home directory, followed by the system default paths, assuming the current shell is `csh`, the following command would be used:

```
setenv TEXINPUTS ~:
```

This is a path of two elements. The first is the user's home directory. The second path element is the empty string, indicating that the system defaults should be searched.

The 'system defaults' as defined here means the strings set in the `Makefile` before compilation, rather than any defaults set in `config.ps` or printer-specific configuration files. This is to prevent path blowup, where more and more directories are added to the path by each level of configuration file.

12. Environment Variables

The `dvips` program reads a certain set of environment variables to configure its operation. The path variables are read after all configuration files are read, so they override values in the configuration files. (The `TEXCONFIG` variable, of course, is read before the configuration files.) The rest are read as needed.

Note that all defaults supplied here are just as supplied in the provided `Makefile`; they will almost certainly have been changed during installation at your particular site.

`HOME` (no default) This environment variable is automatically set by the shell and is used to replace any occurrences of `~` in a path.

`MAKETEXPK` (`MakeTeXPK %n %d %b %m`) This environment variable sets the command to be executed to create a missing font. A %n is replaced by the base name of the font to be created (such as `cmr10`); a %d is replaced by the resultant horizontal resolution of the font; a %b is replaced by the horizontal resolution at which `dvips` is currently generating output, and any %m is replaced by a string that METAFONT can use as the right hand side of an assignment to `mag` to create the desired font at the proper resolution. If a mode for METAFONT is set in a configuration file, that is automatically appended to the command before execution. Note that these substitutions are different than the ones performed on PK paths.

`DVIPSHEADERS` (`.:/usr/lib/tex/ps`) This environment variable determines where to search for header files such as `tex.pro`, font files, arguments to the `-h` option, and such files.

`PRINTER` (no default) This environment variable is read to determine which default printer configuration file to read in. Note that it is the responsibility of the configuration file to send output to the proper print queue, if such functionality is desired.

`TEXFONTS` (`/LocalLibrary/Fonts/TeXFonts/tfm:/usr/lib/tex/fonts/tfm`) This is where `tfm` files are searched for. A `tfm` file only needs to be loaded if the font is a resident (PostScript) font or if for some reason no `pk` file could be found.

`TEXPKS` (`/LocalLibrary/Fonts/TeXFonts/pk:/usr/lib/tex/fonts/pk`) This environment variable is a path on which to search for `pk` fonts. Certain substitutions are performed if a percent sign is found anywhere in the path. See the description of the `P` configuration file option for more information.

`TEXINPUTS` (`.:..:/usr/lib/tex/inputs`) This environment variable is used to find PostScript figures when they are included.

`TEXCONFIG` (`.:/usr/lib/tex/ps`) 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 can thus be redundant.)

`VFFONTS` (`.:/usr/lib/tex/fonts/vf`) This environment variable sets where `dvips` looks for virtual fonts. A correct virtual font path is essential if PostScript fonts are to be used.

13. 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 two arguments to `bop-hook` are the TEX page number and the sequence number of the page in the file; the first page gets zero, the second one, etc. The arguments to `start-hook` are hsize, vsize, mag, hdpi, vdpi, and the name of the `dvi` input file. The procedures must leave these parameters on the stack. The other hooks are not (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 to set printer defaults. For instance, you might have a file `config.dated` that contains just the two lines

```
E echo /bop-hook \{save /Times-Roman findfont 7 scalefont setfont \
                  72 756 moveto \(‘date‘\) show restore\} def >.date
h .date
```

(with no newline following `setfont`); then the command-line option `-Pdated` to `dvips` will print current date and time on the top of each page of output. Note that multiple `-P` options can be used.

14. MS-DOS

The MS-DOS version of `dvips` differs from UNIX in the following ways.

- Path separators are `;` not `:`.
- Directory separators are `\` not `/`.
- The user's initialization file is `dvips.ini` not `.dvipsrc`.
- Pipes to printers are not supported. Output must go to a file.
- `MakeTeXPk` is a batch file. Since MS-DOS has insufficient memory to run both `dvips` and METAFONT at the same time, this batch file will typically write out a set of commands for running METAFONT later. The `maketexp.bat` supplied writes out an `mfjob` file for emTEX.
- `dvidrv` from emTEX can be used to automatically make fonts as follows:

  ```
  dvidrv dvips file.dvi
  ```

  `dvidrv` sets an option `-pj=mfjobfile` for `dvips`, where `mfjobfile` is the name of a temporary `mfjob` file. If there are missing fonts, `dvips` will write this `mfjob` file and then ask:

  ```
  Exit to make missing fonts now (y/n)?
  ```

If you answer yes, `dvips` exits with errorlevel 8 which tells `dvidrv` to call `mfjob` to make the fonts, and then to call `dvips` again. For this to work, `dvidrv`, `dvips`, `mfjob` and `mf` must be located in the `PATH`, and the environment variables for `mfjob` and `mf` set correctly. A font mode must be set with the 'M' option in `config.ps`. If the `-pj` option is set, dvips will not call `MakeTeXPk.bat`.

- Limited emTEX specials. The following ones are supported:

```
\special{em:message xxx}
\special{em:point n}
\special{em:line a[h|v|p],b[h|v|p] [,width]}
\special{em:linewidth width}
\special{em:moveto}
\special{em:lineto}
\special{em:graph xxx[,width[,height]]}
```

The line cut parameters `[h|v|p]` of the `\special{em:line ...}` command are ignored. Lines are ended with a rounded cap. A maximum of 1613 different point numbers are permitted on each page. The `\special{em:graph xxx}` supports `PCX`, `MSP1`, `MSP2` and `BMP` files. The graph file may be scaled by giving an optional width and height (expressed in the same way as TEX dimensions). An example:

```
\special{em:graph scrdump.pcx,100mm,75mm}
```

The program `dvips` can use emTEX font libraries created with the emTEX `fontlib /2` option. If a `pxl` font is found within a font library, `dvips` will complain, and then ignore the `pxl` font.

The font library names and directory names can be specified with this configuration file option.

`L` *path*: The list of `fli` font libraries to search for bitmap `pk` font files is *path*. Fonts are sought first in these libraries and then as single files. Font libraries can be created with emTEX's `fontlib`; the usual extension is `fli`. Directories as well as file names can be entered here, the files will be searched for in all these directories. A directory name must have trailing directory separator (`/` for UNIX, `\` for MS-DOS).

15. Installation

If `dvips` has not already been installed on your system, the following steps are all that are needed.

First update the `Makefile`—in particular, the paths. Everything concerning `dvips` can be adjusted in the `Makefile`. Make sure you set key parameters such as the default resolution, and make sure that the path given for packed pixel files is correct.

Next, check the file name definitions in `MakeTeXPK`. If you don't have METAFONT installed, you cannot use `MakeTeXPK` to automatically generate the fonts; you can, however, modify it to generate `pk` fonts from `gf` fonts if you don't have a full set of `pk` fonts but do have a set of `gf` fonts. If you don't have that, you should probably not install `MakeTeXPK` at all; this will disable automatic font generation.

Now, check the configuration parameters in `config.ps`. You should also update the default resolution here. This file is the system-wide configuration file that will be automatically installed. If you are unsure how much memory your PostScript printer has, print the following file:

```
%!  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; for these systems, a value of one million works well.

Next, run `make`. Everything should compile smoothly. You may need to adjust the compiler options in the `Makefile` if something goes amiss.

Once everything is compiled, run `make install`. After this is done, you may want to create a configuration file for each PostScript printer at your site.

If the font caching is considered a security hole, make the 'cache' directory be something like `/tmp/pks`, and `cron` a job to move the good `pk` files into the real directory. Or simply disable this feature by not installing `MakeTeXPK`.

Don't forget to install the new `vf` files and `tfm` files. Note that the `tfm` files distributed with earlier (pre-5.471) versions of `dvips`, and all versions of other PostScript drivers, are different.

A test program called `test.tex` is provided, so you can easily check that the most important parts of `dvips` have been installed properly.

16. 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.

16.1 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. The classes are:

1	specials
2	paths
4	fonts
8	pages
16	headers
32	font compression
64	files
128	memory

16.2 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.

16.3 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.

16.4 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. You might also try turning off comments on included files with the `-K` option; many spoolers cannot deal with nested documents.

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.

16.5 400 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.

16.6 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.

16.7 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 hex) will probably fail with `dvips`.

16.8 Can't Find Font Files

If `dvips` complains that it cannot find certain font files, it is possible that the paths haven't been set up correctly for your system. Use the debug flags to determine precisely what fonts are being looked for and make sure these match where the fonts are located on your system.

16.9 Can't 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. If `MakeTeXPK` isn't properly edited or isn't installed, an error such as `MakeTeXPK not found` will be printed on the console. The fix is to talk to the person who installed `dvips` and have them fix this.

If METAFONT isn't found when `MakeTeXPK` is running, make sure it is installed on your system. The person who installed TEX should be able to install METAFONT easily.

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 `local.mf` (on the NeXT, `next.mf`; on the Amiga, `amiga.mf`), type the following command (assuming `csh` under UNIX):

```
localhost> inimf "plain; input local; dump"
```

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

Note that a preloaded `cmbase.base` should never be used when creating fonts, and a program such as `cmmf` should never exist on the system. The macros defined in `cmbase` will break fonts that do not use `cmbase`; such fonts include the LaTEX fonts. Loading the `cmbase` macros when they are needed is done automatically and takes less than a second—an insignificant fraction of the total run time of METAFONT for a font, especially when the possibility of generating incorrect fonts is taken into account. If you create the LaTEX font `circle10`, for instance, with the `cmbase` macros loaded, the characters will have incorrect widths.

17. 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.

17.1 The 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 lose 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 delimiter 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.

17.2 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 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.

17.3 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 insure that TEX is in horizontal mode before the first special command is issued; this can be done by prefixing the color command with `\leavevmode`.

17.4 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.

17.5 Configuring dvips for Color Devices

To configure dvips for a particular color device you need to fine tune the color parameters to match your device's 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.

17.6 Protecting Regions From Spurious Colors

Because color is defined via TEX's `\special` command, it cannot be sensitive to the output routine or certain regions of the page like the header and footer. Consequently, these regions need to be protected from spurious color changes (particularly when local colors spread across page breaks).

Users need to be aware of the possibility of certain regions getting unwanted or unpredicted colors. Headers and footers are most worrisome so style designers who want to use color should keep this in mind.

One particular region of text that gets spurious color effects is labels in list environments. For instance, because of the way list items are defined in standard LaTEX, the bullet for items that start with a different color also gets drawn in that color.

To give the user a simple mechanism to solve this problem (and other unforeseen effects of this type) one other special macro is automatically defined. This macro is called `\globalColor`. It is actually a *local* color macro and so takes a single argument. But the color effect it produces is always the same as that set by the *last* `\textColor` or `\textColorName` command. In effect, when a `\textColorName` command is called, `\globalColor` gets a new definition equivalent to the local `\ColorName` macro. For example, when the default is black, `\globalColor=\Black` and when `\textGreen` appears, `\globalColor=\Green`. This special macro can then be used to protect sensitive regions of the text.

For example, in LaTEX files, one might make sure that the header and footers have `\globalColor` wrapping their contents. In this way, they will inherit the current active root (unnested) color state.

17.7 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.

4 Using TEXview

It is the TEXview program that really makes NeXTTEX shine, and it is the capabilities of the NeXT machine that allow TEXview to work so well. The PostScript-based windowing system makes it easy to preview your document—complete with all the PostScript graphics as they will appear on paper—quickly and painlessly. Unix multitasking and a windowing environment make it possible to view your document as you edit and run TEX, without ever having to exit any of the programs.

1. Basic Operation

There are three ways to start up TEXview. The first is to just double-click on the application icon in the Browser or on the dock. After a few seconds, the program will be launched.

Alternatively, you can use the NeXT `open` command. This unique command examines the suffix of the given file and automatically invokes the appropriate application to display the file. With the `open` command, there is no need to remember the names of all the various applications required to view files—you simply 'open' each file. TEXview responds to `open` for files with the extensions `.dvi` or `.tex`.

If you prefer an iconic approach, you can simply double-click on a `tex` or `dvi` file, and TEXview will automatically be launched and display that file. If you want `tex` files to be opened by an editor rather than by TEXview, you can do this with the Inspector tool in the Workspace browser after selecting any file with the extension `.tex`.

The third way is useful if you want to run TEX within another environment, such as `emacs`; simply use the `-V` option to TEX when you process a file, and TEXview will be launched and accept the file that TEX is currently working on.

Whenever TEXview opens a `tex` or `dvi` file, it switches its current directory to the directory that file resides in. This makes it easy for TEXview to find the appropriate graphics and other necessary files, without having to constantly change environment variables.

The entire functionality of TEXview is available through the various buttons and menu options, so feel free to browse through the program now, trying things out. The remainder of this document gives details on certain operations and various defaults you can set and

use. It is recommended that you launch TEXview now and try out the controls as they are described.

One thing to remember about TEXview is that it takes longer to start up than it does to display a page or go to a different file. It is usually smarter to 'hide' the program than it is to quit it, since when you need it next it will be available and probably have all of the fonts loaded and ready to go. For example, the time required to launch and load a short sample file at one installation was measured at nine seconds, while the time to display a new file from an already launched TEXview was under a second. The more fonts that are used in a document, the greater this disparity is. So try to get in the habit of using 'hide' rather than 'quit'. This is true of other NeXT applications as well.

All of the details on including PostScript graphics and fonts are given in the chapter on `dvips`.

2. Windows

The TEXview program uses a variety of windows. At various times you may want to use only one or a few of these windows; you can independently close or open any of them. They each have very different characteristics.

2.1 The Preview Window

The main window is the preview window itself; this is the large window with scrollbars on the left side and bottom. It is the window that displays what the printed pages will look like.

In addition to the scrollers that are used to move around on the page, the Preview window also contains a zoom button that can be clicked to select the zoomed or unzoomed resolution. More details on the zoom resolutions will be given later. In addition, it also contains a page number field that can be edited. To go to a particular page number, just type that number and hit carriage return.

The page itself can be click-dragged with the mouse; this is a way to change the visible region of the displayed sheet without using the scroll bars.

When this window is the key window, menu items can be selected by hitting just the corresponding key, without pressing the command qualifier. For instance, to re-run TEX on the current document, you can just hit 't' rather than command-'t'. This makes use of TEXview a little easier on the small fingers of the hand. In addition, a number of additional keystroke commands are supported. The four arrow keys can be used to move around on the page and from page to page. Normally, the up and down arrow keys move the page one screenful up or down, and the left and right arrow keys move back and forward a page, respectively. The arrow keys can be 'qualified' with shift and alternate to reduce the amount

of scroll. In addition, the left and right arrow keys move the window around on the page when they are so qualified.

The space key also advances to the next page, and the backspace and delete key go back a single page.

The following table lists the most commonly used keyboard commands in TEXview:

Keyboard Shortcuts	
F	First page
N	Next page
B	Previous page
L	Last page
up arrow	Up
down arrow	Down
right arrow	Next page
left arrow	Previous page
space	Section forward
delete	Section backward
z	Zoom
u	Unzoom
t	ReTEX
l	ReLaTEX
s	ReSliTEX
R	Re-Open
o	Open new file
c	Close file
h	Hide
Q	Quit

You can measure distances on the preview window by clicking on the page. The position at which you clicked and the distance from the most recent click is displayed on the command window; see the command window for some more details.

The preview window can be brought to front with command-shift-P. It remains visible when the application is not active, and its size and position are considered part of Preferences. It can become the key window.

2.2 The TEX Window

TEXview uses the TEX window to display the results of a run of TEX or LaTEX. This window provides a scrollable text object containing the terminal session of the run. You can type characters into the window in response to queries much like you would when you run TEX from a terminal or shell. In particular, to interrupt TEX, you can type control-C. Note that the window must be activated before it will accept keystrokes.

As TEX runs and processes pages, they will become available for viewing in the main preview window.

Note that it is nice to use a line of the form

`%&` *formatname*

when using TEXview, so it knows what format to use when automatically running TEX. For plain TEX, the *formatname* should be `plain` or just `tex`; for LATEX it should be `lplain` or `latex`, and for SLITEX it should be `splain` or `slitex`. For your own custom format files, it should be the name of the format file in `/usr/lib/tex/formats` and also the name of the executable under which TEX is run.

The TEX window can be brought to front with command-shift-T. This window disappears when the application is not active and its size, position, and visibility are considered part of Preferences. It can become the key window.

2.3 The Console Window

For printing and faxing, TEXview uses the `dvips` program. When TEXview needs a font at a size that is not currently available, it runs METAFONT. The results of these commands are displayed in the console window. As long as the command in the console window is still executing, TEXview will not respond to other commands.

The console window can be brought to the front with command-shift-C. This window disappears when the application is not active and its size, position, and visibility are considered part of Preferences. It can become the key window, even though it does not accept keystrokes.

2.4 The Command Window

Some people like the clean and elegant preview window, along with menus and keyboard shortcuts, for their operation of TEXview. Others prefer a more helpful interface with buttons for various operations. The TEXview command window provides this functionality. In addition, some seldom-needed operations can be performed from the command window.

The command window contains buttons for the operations first page, last page, next page, and previous page. It also has buttons to open a new file, re-open the current file, and run TEX with the current (last-used, default) format, TEX with the `plain` macros, LATEX, TEX with a custom format specified on the Preferences panel, or BIBTEX on the current file. Selecting one of `Plain`, `LaTeX`, or `Custom` makes that the current default format.

The command window also includes the zoom resolution popup that displays the current resolution at which the sheet is being displayed, as well as allowing the selection of a new resolution for the current (zoom or unzoom) mode.

The position at which a user clicks on the sheet is displayed in the command window, and there is a button that can be clicked to change the units that the measurements are given in.

The command window can be brought to the front with command-shift-X. This window remains visible when the application is not active, and its position and visibility are considered part of Preferences. It cannot become the key window. This makes it possible to put the command window somewhere convenient on the screen while you are working in your editor, and then with a single keystroke on the command window, re-run TEX on your file and bring the TEXview window to the front. This can also be done with the services menu, which we will discuss later.

2.5 The Preferences Window

The Preferences window contains a few controls for setting some options in TEXview. In addition, by bringing up the Preferences window and selecting 'Save', the entire configuration of TEXview is saved and made the default for future sessions. This configuration includes the size, location, and visibility of many of the TEXview windows, the current zoom and unzoom resolutions, the current default and custom format files for TEX, and much more.

If you do not select the Save button on the Preferences window, no Preferences will be saved. This means that after selecting a new zoom resolution, or after moving the TEX window, if you want the changes to persist to the next session, you must select the Save buttons on the Preferences window.

The Preferences window allows you to set the sheet size to be assumed when displaying a `dvi` page. By default, this is set to a standard 8.5 by 11 sheet, with the TEX origin one inch from the top and left hand corner of the page. If you are using different paper, you can easily edit the values in the Preferences panel. Normally the information in this panel should be set from a `papersize` special within the TEX document itself; see the chapter on `dvips` for more information on this.

A technique that is sometimes useful is to edit the sheet size in TEXview to eliminate the displayed margins. For instance, if I am writing a paper with one inch margins on all sides, I will often tell TEXview that I really have a sheet of paper that is only 7 by 9.5, with the origin at one quarter inch from the left and the top. This effectively trims three quarters of an inch from all four margins when TEXview is displaying it, so I can more effectively use the scrollers. Doing this has no effect on printing the document or on the `dvi` file that is generated.

Another item on the preferences window selects whether to use `MakeTeXPK` to generate a missing font when it is needed by TEXview, or to scale an existing font. The default is to scale existing fonts. Using automatic font generation slows down TEXview significantly, since generating a missing font can take up to two minutes of delay during which the program does not respond to any user actions.

If a bitmap font is scaled, the display on the screen will be somewhat ugly. To help solve this, the name of the desired font is written to the file

```
/LocalLibrary/Fonts/TeXFonts/pk/NeededFonts
```

If this file is executed, all fonts that needed to be scaled will be generated, so the next time that file is viewed, the screen representation will look much better. It is often useful to `cron` a job to execute and delete this batch file each night. If this is done, make sure that the batch job is run as a normal user and not superuser, since anyone can append commands to this file.

In addition, the Preferences window contains a button specifying whether included graphics should be displayed (for accurate presentation) or 'hidden' with a gray box (for quicker display.) Only for the most complicated graphics is it necessary to choose this button, because the NeXT displays PostScript extremely quickly.

The Preferences window also includes text fields that allow you to specify the default format file to use when re-TEX'ing a file and the 'custom' format to use when the Custom button or the Custom menu item is selected.

This window disappears when the application is not active, and it is never visible when TEXview is launched.

2.6 Other Windows

TEXview also includes a simple window with some suggestions on how to use the program.

There is also an info window that contains the version number of the application. Please let us know the version number of TEXview if you report a bug.

These windows disappear when the application is not active, and they are never visible when TEXview is launched.

3. Zoom and Unzoom Resolutions

TEX generally uses bitmapped fonts rendered by METAFONT. Rendering these fonts is slow, and the bitmaps take up precious disk space. To help make TEXview run faster, and to reduce the amount of disk space required by the program, TEXview makes it easy for the user to select two common sizes to view documents at.

The command window includes a popup that allows you to select one of a dozen or so resolutions at which to view the document; they range from 69 to 400. The number is in dots per inch, where per inch here means per inch of the document, not per inch of

the screen. (This latter is somewhat fixed.) The larger number you select, the larger your document will be magnified. Selecting either 300 or 400 can cause your document to take several seconds to render due to the amount of memory required.

Just this last control suffices to select any of a number of resolutions. But using it is awkward; you have to click the mouse, make a selection, and release the mouse. In addition, each time a new resolution is selected for a particular document, fonts need to be loaded and perhaps scaled; this takes time and memory. The `zoom` button at the bottom of the TEXview window solves both of these problems.

Typically, TEXview is used to display and scroll through a document at a single resolution. Occasionally, when more detail is desired, this view is 'zoomed' to a higher resolution, so more detail can be seen. So, two resolutions are commonly used.

The `zoom` button switches between two stored resolutions, each of which can be changed with the popup button. By selecting two resolutions at which you wish to work, and using the `zoom` button rather than the popup to select them, you limit the number of fonts that need to be loaded and scaled and the amount of memory that TEXview uses, thus yielding faster operation of the program.

Note that there is no real difference between 'zoom' and 'unzoom'; there is no requirement that the 'zoom' resolution be higher than the 'unzoom' resolution. Each can be set arbitrarily.

Let us say that you want to set your zoom resolution to 144 dots per inch and your unzoom resolution to 69. You would first make sure that the Zoom button at the bottom of the main Preview window is not checked. Then, you would bring up the Command window with command-shift-X (or with the `Windows/Command` menu item). You would use the popup button to select 69. Then, after waiting for TEXview to re-render the page, you would click the Zoom button, so that it is set. Then, going back to the Command window, you would use the popup button to select 144. After letting the page be rendered again, you would go to the Preferences panel with the `Info/Preferences` menu option and select `Save Preferences` to make sure the changes persisted to the next TEXview session. Now, in normal usage, you would only need to click the Zoom button on and off; you would not need to use the popup button on the Command window.

4. Menu Options

TEXview comes with a wide variety of menu operations. It follows most NeXT conventions, so little needs to be said about them. In this section, we discuss some of the limitations of the options, and how the use of the menu items differs from convention.

Most menu options and subitems have keyboard shortcuts. These are displayed on the menu itself as a single letter at the end of the menu item, and they are executed by holding down one of the command keys and pressing the appropriate letter. The keyboard shortcuts that are upper case require you to hold down both one of the command keys and one of the

shift keys on the keyboard. If the preview window is active, the keyboard shortcuts can be accessed without holding down the command key.

The `Document` submenu only contains two entries: `Open` and `Reopen`. Note that TEXview can only display one file at a time in its single window, so selecting `Open` will close the file currently being displayed. A future version may be able to display multiple files. The `Reopen` entry will open the current file again, in case the disk file has changed.

The `Edit` and `Font` submenus only apply to the Command, TEX, and Hints windows; you cannot cut, copy, or paste to the main preview window. This is because TEX is a command-oriented system that compiles an ASCII file into a document; figuring out the relevant changes to make to the original input file in order to implement the cut or paste is an extremely difficult operation.

The `Compile` menu lists the commands that are performed in the TEX window when that menu item is chosen. The string `%s` is replaced by the current file name, with no extension. The string `%L` is replaced by the default, or last, format file name. The string `%C` is replaced by the custom format file name. (Note that the `plain` format is actually called `tex`, since the command name of the TEX program and the format file it loads have the same base name.) Choosing just `TEX` uses the most recent, or default format; choosing one of the others makes that the current default format.

5. Printing from TEXview

Printing from TEXview is similar to printing from other NeXT applications. The print panel brought up by the `Print` menu option is the same as used with other applications, except that a few more controls are added. Printing itself is done through the `dvips` program rather than more typical NeXT methods. This is required to properly handle TEX bitmap fonts and to generate reasonably small output PostScript files.

The additional controls include ones to set collated copies, reversed output and compressed fonts. Please refer to the previous chapter for information on these. Note that 'reversed output' just means that `dvips` reverses the order of the pages; because the NeXT printer prints face-up, the last page of the document must be printed first. If you don't select 'reverse', then the spooler does this reversal. The `dvips` program can do this reversal much more quickly than the spooler can, since it just needs to generate the pages in reverse order. In either case, the printed output will come out with the first page on top.

You can also select a custom resolution. This is useful for generating typesetter output to a file, for instance. Simply select the `Custom resolution` button and enter your desired resolution in the gadget.

The `Wait on dvips` button selects whether to wait until `dvips` completes before continuing TEXview operation, or to just let `dvips` run in the background.

The `Preview` button on the print panel does nothing, since TEXview is itself a previewer.

6. Services

NeXTTEX offers services to other applications. These services are available whenever the active application has data of the correct type to send. The services can be accessed through the `Services/TeXview` menu of an application, and are only active when they are not 'ghosted'. The services are not available to those applications that do not make the correct types available, or accept the correct types back. It is possible to change the services that TEXview accepts; we will discuss that after we describe the services that TEXview offers as shipped.

6.1 Predefined Services

TEXview as distributed offers seven services. They are `TeX (plain)`, `LaTeX`, `SliTeX`, `TeX (custom)`, `ReTeX`, `Reopen dvi`, and `TeX EQ->.EPS`. We will describe each in turn.

The first four require a file name (`NXFilenamePboardType`) as its argument and returns nothing. Most editors (including `Edit`) support this file type; it is typically the current file being edited. These services passe the current file name to TEXview to be opened. If the file has an extension of `.tex`, it is TEX'ed and then displayed. Otherwise, it is taken as the name of a `.dvi` file and directly displayed. With this Service, you can, with a single menu selection, send the name of the current file to TEXview for viewing. In addition, the current TEX format is set as specified, either `plain`, `latex`, `slitex`, or the current TEXview `custom` format.

Most NeXT applications do not enable services that neither take nor return data types, so each service must either take or return some data. With TEX, most editing of non-trivial documents takes place in a sub-file. For instance, in a dissertation, editing is normally done in the individual chapter files, while it is the top-level dissertation file that we want TEX'ed. For this reason, the `Reopen dvi` and `ReTeX` commands take but ignore data of type `NXFilenamePboardType`. They simply re-process the current file being displayed by TEXview, in the latter case, using the most recent format specified.

The `Reopen dvi` command just reopens the `dvi` file. It is useful if you are running TEX from within some other application, such as `emacs`. The `ReTeX` command, on the other hand, re-runs TEX from within TEXview and displays the result as it becomes available. If you use this service, it is important that you also adopt the `%&` convention for specifying the appropriate format file for TEX to use, or else that you have already chosen the appropriate format type by selecting one of the first four services.

These six services provide the most common functions needed in the typical edit-compile-view loop. Typically, the first time you process a file, you will use one of the first four commands to open the file and select the appropriate format. From then on, you will use either `ReTeX` (to run TEX from within TEXview) or `Reopen dvi` (if you have run TEX through a different application) to view the changed file.

Another service, `TeX EQ->.EPS`, takes as input data some ASCII clip, of type `NXAsciiPboardType`, and treats it as a TEX equation to be typeset. It runs TEX and `dvips -E` over the data, and responds with a PostScript clip of type `NXPostScriptPboardType`. For instance, bring up Edit. Within Edit, type the following line (exactly):

```
$$\sum_{i=0}^\infty {x^i\over i!}=e^x$$
```

Now select the entire line with the mouse, and cut it, and then paste it twice. Now select one of the instances with the mouse, and then choose the menu entry `Services/TeXview/TeX EQ->.EPS`. After a few seconds, perhaps a little longer, the equation should appear in the `Edit` window! It will not be very readable, because TEX uses bitmapped fonts, but then select Print from within `Edit` and the output should look fine.

The equations will look much better—indeed, very good—if you use PostScript versions of the CM fonts, such as the ones available from Blue Sky Research. Their phone number is (503) 222–9571. These fonts are commercial and are not terribly cheap, but for many uses they are very, very nice.

6.2 User-Defined Services

It is possible to extend the service support in TEXview with your own services. For instance, if your editor does not support the `NXFilenamePboardType` but does support commands that neither send nor return data, you can add support for such a service.

Start with the file `TeXview.service` in `/usr/lib/tex/source/misc`. Make a copy of this file. This entire file looks (in part) like:

```
Message:  serverReopen
Port:  TeXview
Send Type:  NXFilenamePboardType
Menu Item:  TeXview/Reopen dvi
KeyEquivalent:  R

Message:  serverCommand
Port:  TeXview
User Data:  Send Type:  NXFilenamePboardType
Menu Item:  TeXview/ReTeX

Message:  serverMakeEquation
Port:  TeXview
Send Type:  NXAsciiPboardType
Return Type:  NXPostScriptPboardType
Menu Item:  TeXview/TeX EQ->.EPS
ActivateProvider:  NO
DeactivateRequestor:  NO
```

Note how it defines the four services we described above. Simply copy the `TeXview/ReTeX` service to the end of the file and delete the Send Type line. Also change the menu item name (perhaps to `TeXview/ReTeX2`). Save this new file as `TeXview.service` in the directory `/LocalLibrary/Services` (you may have to create this directory if it doesn't already exist.)

Now exit TEXview if it is running, and re-launch it. You should see an additional service entry in the `Services/TeXview` menu while in your editor. That's all it takes! (You will also probably have to deselect and then reselect the Services menu entry in your editor.)

7. PostScript Graphics and Fonts

Including PostScript graphics and using PostScript fonts works exactly as described in the previous chapter on `dvips`; please refer there for details. Note that TEXview does not handle the `bop-hook`, `eop-hook`, `start-hook` and `end-hook` extensions. In addition, any header files loaded are loaded once, permanently, so any effects will take place for all subsequently loaded `dvi` files. Since header files typically only define macros, rather than executing anything, this should seldom cause difficulty. When it does, it is necessary to exit TEXview and start it up again to return to a clean state.

8. Defaults Variables

The NeXT machine has a powerful system for setting and viewing application defaults. These variables are set automatically by the Preferences window in TEXview, but the system is also accessible from the `dread` and `dwrite` commands; please refer to NeXT documentation on these programs. The variables recognized and used by TEXview, their defaults, and an explanation of each follow.

`commandposition` (none) This variable holds the default window position and activation status of the command window.

`compressed` (`false`) This variable sets whether compressed fonts are used by default in printing through `dvips`; it has no effect on TEXview itself.

`consoleposition` (none) This variable holds the default window position, size, and activation status of the console window.

`customdpi` (`1270`) This variable sets the value of the default custom resolution, if custom resolution is selected. The default value is a common value used when driving certain Linotronic typesetters.

`customformat` (`amstex`) This variable holds the default 'custom' format file name for TEX. This format is chosen when you select the custom command button or the Custom menu item; it can be changed in the Preferences window.

`defaultfont` (none) This variable holds the default font for the TEX and console windows.

`defaultformat` (`tex`) This variable holds the default format file name for TEX.

`debug` (`0`) This variable should be seldom used. If set, it should be set to a value from 0 to 9, with a higher number yielding more detailed debugging information. The debugging information is written to standard output, so make sure you invoke TEXview from a shell.

`generatefonts` (`false`) This variable controls whether to generate fonts with `MakeTeXPK`, or to scale other bitmapped fonts to fit.

`hmarg` (`1`) This variable is the horizontal offset of TEX's origin point on a sheet of paper, in inches.

`hsize` (`8.5`) This variable sets the horizontal size of the default TEXview sheet, in inches.

`mocked` (`false`) This variable determines whether included PostScript graphics are rendered or 'faked' with a gray box.

`paranoia` (`0`) This variable holds the current paranoia level. If it is equal to 1, then the TEXview console is not used, and the browser's console window is used instead. This might prevent TEXview from hanging in some circumstances.

`reversed` (`true`) Determines whether `dvips` should be told to reverse the pages when printing. If `dvips` doesn't do it, the spooler will, and `dvips` can do it faster. Reversing the order of the pages is necessary because the NeXT printer stacks the output face up.

`sync` (`true`) If this flag is true, then TEXview will wait for `dvips` to complete before returning to the user from the print panel; otherwise, the `dvips` process is spawned and executed concurrently with TEXview.

`texposition` (none) This variable holds the default window position, size, and activation status of the TEX window.

`unzoomres` (`91.287102`) This variable sets the default unzoom resolution for TEXview. Only certain magic values are acceptable; use TEXview to change this rather than `dwrite`.

`vmarg` (`1`) This variable is the vertical offset of TEX's origin point on a sheet of paper, in inches.

`vsize` (`11`) This variable sets the vertical size of the default TEXview sheet, in inches.

`windowposition` (none) This variable holds the default window position and size of the main preview window.

`zoomres` (`120`) This variable sets the default zoom resolution for TEXview. Only certain magic values are acceptable; use TEXview to change this rather than `dwrite`.

9. Environment Variables

TEXview uses some of the same environment variables as `dvips`, and they are interpreted in the same way, so refer to the previous chapter for more information.

These environment variables must be set in your `.cshrc` file in your home directory, since this is the only file that TEXview examines for environment variables. Even if you use a different shell, make sure you update this file for `csh` if you want TEXview to find the appropriate files. Also, when TEXview runs TEX, it runs it under a `csh` so it is important for TEX as well.

`HOME` (no default) This environment variable is automatically set by the shell and is used to replace any occurrences of `~` in a path.

`MAKETEXPK` (`MakeTeXPK %n %d %b %m`) This environment variable sets the command to be executed to create a missing font. A %n is replaced by the base name of the font to be created (such as `cmr10`); a %d is replaced by the resultant horizontal resolution of the font; a %b is replaced by the horizontal resolution at which `dvips` is currently generating output, and any %m is replaced by a string that METAFONT can use as the right hand side of an assignment to `mag` to create the desired font at the proper resolution. If a mode for METAFONT is set in a configuration file, that is automatically appended to the command before execution.

`TEXCONFIG` (`.:/usr/lib/tex/ps`) 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 can thus be redundant.)

`TEXFONTS` (`.:/LocalLibrary/Fonts/TeXFonts/tfm:/usr/lib/tex/fonts/tfm`) This is where `tfm` files are searched for. A `tfm` file only needs to be loaded if the font is a resident (PostScript) font or if for some reason no `pk` file could be found.

`TEXINPUTS` (`.:..:/usr/lib/tex/inputs`) This environment variable is used to find PostScript figures when they are included.

`TEXPKS` (`.:/LocalLibrary/Fonts/TeXFonts/pk:/usr/lib/tex/fonts/pk`) This environment variable is a path to search for `pk` fonts on. Certain substitutions are performed if a percent sign is found anywhere in the path; see above for a description of the substitutions performed.

`VFFONTS` (`.:/usr/lib/tex/fonts/vf`) This environment variable sets where `dvips` looks for virtual fonts. A correct virtual font path is essential if PostScript fonts are to be used.

5 Additional Utilities

This chapter gives some brief instructions for various utilities supplied with NeXTTEX. Many of these utilities are mentioned in other chapters; some are not. This chapter does not include descriptions of some of the programs more extensively described in the other chapters.

The utilities are listed in alphabetical order. For more information on various file formats or more extensive documentation on any of these programs, write Radical Eye Software. (If every program included were fully documented, it would take an entire row of a bookshelf to hold the documentation.)

1. dvitype

This utility converts a TEX `dvi` file to a human-readable (and usually very verbose) form. Usage is

```
localhost> dvitype foo.dvi
```

The program will engage you in a dialog concerning various options, such as the assumed device resolution and how verbose the output should be. The actual output itself is always written to a file called `dvitype.out`, and it may well be extremely large, so make sure you have space for it. In verbose mode, the output is often more than 40 times larger than the `dvi` file.

2. gftodvi

This program makes proof sheets for fonts, assuming that a `gf` file was created with `proof` mode on. Documentation is given in *The METAFONTbook*, Appendix H.

To create proofs, you must have a gray font at the appropriate resolution. Note that gray fonts are not device independent; the `tfm` file for a gray font varies depending on the device that is to be driven. Thus, typically you will create `gray.tfm` and `gray.pk` for the particular device you wish to drive. Since `gftodvi` uses the `tfm` file to create its `dvi` file, you cannot run `gftodvi` until you have the gray font built.

As an example, to build a gray font for a standard 300 dpi device, you would create a file with the single line

```
input grayf
```

and then you would run the commands

```
localhost> mf "\\mode:=imagen; input gray"
localhost> cp gray.tfm /usr/lib/tex/fonts/tfm
localhost> gftopk gray.300gf /usr/lib/tex/fonts/pk-cache/gray.300pk
```

to create the gray font. (For a 180 dpi device, for instance, you would substitute `NEC` and `180` for `imagen` and `300` above.) Then, as an example of a simple proof sheet, type

```
localhost> mf cmr10
```

and let it run. (The default is `proof` mode.) This will take a while and create a very large `gf` file, but with full proof marks. If you only want an example, you can interrupt it after a few characters; the `gf` file it will write will be well-formed if incomplete. Now, run `gftodvi` with

```
localhost> gftodvi cmr10.2602gf
```

and let that grind for a while. (The `dvi` file will be even larger than the `gf` file.) Now, print the file. You will probably want to print no more than a couple of pages if you are just looking; an entire 128 character font is a pretty hefty stack of paper. (Previewing it will not work since the `gray` font is device dependent.)

3. gftopk

The `gftopk` program converts a `gf` file to a `pk` file. Usage is

```
localhost> gftopk cmr10.300gf cmr10.300pk
```

for example. If the aspect ratio is not exactly 1, a warning will be printed; this is perfectly normal and nothing to worry about.

4. gftype

This program converts a `gf` file to a human-readable representation, and checks the `gf` file in the process. Usage is

```
localhost> gftype cmr10.300gf
```

You can add a `-i` switch before the file name to print out the pixel images of each character, and a `-m` switch to turn on `gf` format mnemonics.

5. mft

The `mft` program converts a METAFONT source that obeys certain conventions into a nice TEX file for printing. This allows very nice pretty-printing of METAFONT source. Usage is

```
localhost> mft foo.mf
```

6. patgen

This program takes a list of hyphenated words and generates hyphenation patterns for use by TEX. Usage is

```
localhost> patgen dictfile patfile outfile
```

7. pktogf

This program converts a `pk` file to a `gf` file, essentially undoing the compression of `gftopk`. No information is lost in either direction, although the files will be different because of embedded comments. The usage is

```
localhost> pktogf cmr10.300pk cmr10.300gf
```

In general, this program will very seldom be needed.

8. pktype

This program prints out a `pk` file in a human-readable form. It is used by

```
localhost> pktype cmr10.300pk
```

The output is written to the screen or can be redirected into a file.

9. pltotf

This program takes an ASCII version of a `tfm` file and compiles the `tfm` file. Usage is

```
localhost> pltotf cmr10.pk cmr10.tfm
```

The format is a simple Lisp-like language, except that the indentation level at which parentheses are found is expected to be consistent. For an example `pl` file, run

```
localhost> tftopl cmr10.tfm cmr10.pl
```

10. tftopl

This program converts a TEX font metric, or `tfm`, file, to a property list, or `pl`, file. This and companion `pltotf` are useful to change the kern data in a font or to do other tricks. Usage is

```
localhost> tftopl cmr10.tfm cmr10.pl
```

11. vftovp

This program is much like `tftopl`, but it turns a virtual font file (`vf`) into a virtual property list file (`vpl`). It must read not only the `vf` file but also the corresponding `tfm` file, so run it with

```
localhost> vftovp foo.vf foo.tfm foo.vpl
```

12. vptovf

This program takes a given `vpl` file and converts it to the binary `vf` and `tfm` files. Usage is

```
localhost> vptovf foo.vpl foo.vf foo.tfm
```

6 Using PostScript Fonts

This chapter gives some more information on using PostScript fonts with the NeXT in TEX. The information in the chapter on `dvips` was specific to `dvips`; this chapter will be specific to the NeXT.

There are many advantages of PostScript fonts. They scale quickly and easily compared to METAFONT fonts. They are very portable. There is a wide variety of styles available. They are a publishing standard. There are disadvantages as well. They can be larger and slower than bitmapped fonts. They do not always have all of the characters that TEX wants. They are less standard in the TEX community than METAFONT fonts. And when something goes wrong, it can be very difficult to determine what the problem is.

PostScript fonts can be installed for use with just NeXT TEX, or for general use by the NeXT software, or for both. One does not imply the other; if you have installed a font on the NeXT, there are additional steps you must perform to get it to work with NeXT TEX; conversely, if you have installed a font for use with NeXT TEX, you may not have done everything necessary to get it to work for other applications on the NeXT.

This chapter will first describe font files in general. Then, it will present the requirements for installing a font on the NeXT for use by various application software. Then, we will consider the installation requirements for use of the font with NeXT TEX. Finally, we will present three case studies of installing PostScript fonts on the NeXT for use with TEX.

I recommend you read the entire chapter, even if a section may not apply to your particular situation.

1. Font Files

PostScript fonts are generally supplied in two files—a font program (often with the extension `.pfb` or `.pfa`, or, with no extension) and font metrics (with the extension `.afm` or `.pfm`.) The first file is loaded by the PostScript interpreter and contains the actual instructions for rendering the fonts on a character by character basis; the second file contains information on character widths, heights, depths, the character set encoding, and various other information that is needed by the application that wants to use the font.

The font program can have several different characteristics. First, it can be either a Type 1 or a Type 3 font (and even other types are possible.) A Type 1 font has full hinting and will usually look much better at resolutions below about 1000 dpi than the equivalent unhinted Type 3 font; on the other hand, a Type 1 font can only use a very narrow subset of the PostScript language, whereas a Type 3 font can use the full language. For the purposes of installation, there is no difference between a Type 1 and a Type 3 font.

Secondly, a font can be encrypted or not. If it is, the font will be mostly binary or hexadecimal data. If it isn't, the font might still be mostly binary or hexadecimal data, but it is often plain PostScript text. In any case, whether a font is encrypted or not does not matter for installation.

Third, a font can be stored in ASCII format (often with the extension `.pfa`), or it can be stored in compact binary format (often with the extension `.pfb`). In general, fonts for Unix come in ASCII and fonts for MS-DOS and other platforms come in binary. The NeXT can only work with ASCII fonts; NeXT TEX, on the other hand, can handle either ASCII fonts or MS-DOS style binary fonts. (Macintosh fonts may or may not work, depending on how the resource fork is mapped into the flat file format of Unix.)

To tell whether a font is in binary or ASCII format, just examine the first byte. If the first byte has the value 128, then it is a binary font file; otherwise, it is probably an ASCII file. (This is the method `dvips` and `TEXview` use to determine whether the font is binary or not.) Another way is to simply type `file` followed by the font file name; if 'data' is returned, then the font is probably binary; if 'shell commands' or 'ASCII text' is returned, then the file is probably ASCII.

If you plan to install the font for the NeXT, and the font program as you get it is in a binary format, you must convert it to ASCII form before you can use it for NeXT software other than TEX. The program `undos` is supplied (in both binary and source form) in the directory `/usr/lib/tex/src/misc`; to convert the binary font program `Foo.pfb` to the ASCII font program `Foo.pfa`, just type

```
localhost> /usr/lib/tex/src/misc/undos <Foo.pfb >Foo.pfa
```

Now, simply use the `Foo.pfa` file on the NeXT.

The font file must not contain non-ASCII characters, such as control-D or control-Z. If your font doesn't appear to work, check the ASCII font program file (especially the beginning and end) for these and other control characters. If you find such characters, complain loudly to your font vendor; they have no business in a PostScript file of any sort, especially a font file.

Another potential problem is that the font file is one that has an 'exitserver' command in it to permanently download the font to the printer. This is not necessary and will probably cause the font to fail. (Note that many fonts may actually have the command in the font, but not execute it, so mere presence of the word 'exitserver' in the font file does not necessarily mean that the font has this problem.)

Note that ASCII font files tend to be about twice as large as binary font files, so a moderate collection of PostScript fonts can take a lot of disk space. It is to be hoped that the NeXT will soon support the binary format so this space can be reclaimed.

The font metric files can be either `.afm` files (an ASCII format), or `.pfm` files (a binary format common for MS-DOS programs), or both. Both NeXT TEX and the NeXT in general require `.afm` files, and no program to convert `.pfm` files to `.afm` files is supplied. If you only received a `.pfm` file, contact your font vendor for the `.afm` file.

2. Installing Fonts for the NeXT

Unfortunately, installing new PostScript fonts on the NeXT is still not an entirely foolproof operation. This section of the manual may not be entirely accurate, and it certainly does not cover every case.

In order for the NeXT to use a font, it must be able to find an `afm` file and it must be able to find a font program with the same name as the actual font. These two files must reside in a special directory with the name of the font followed by `.font`. For instance, the `Times-Roman` font is stored in the directory `Times-Roman.font`, which contains the files `Times-Roman.afm` (the font metric information) and `Times-Roman`, the font 'binary'.

These font directories (here, we refer to the actual disk directories, such as `Times-Roman.font`) must be located in one of `/NextLibrary/Fonts`, `/LocalLibrary/Fonts`, or `~/Library/Fonts`. You might browse these directories to see what fonts you currently have in your system.

In addition to these directories, the NeXT also uses some 'hidden' files to keep track of these fonts. These files are called `.fontlist`, `.afmcache`, and `.fontdirectory`. Normally, the NeXT will maintain these files itself; if it fails, or the fonts don't appear in font panels, see the man page for `buildafmdir`. And if it doesn't work, contact NeXT for information.

3. Installing Fonts for NeXT TEX

To install fonts for NeXT TEX, simply follow the instructions given in the chapter on `dvips`. Please read that chapter for more information; in general, the information in that chapter is assumed in the following discussion.

You will need an `.afm` file for the font. You will need to determine a name that TEX can use to access the font; for standard Adobe fonts, this can be done with the auxiliary file `adobe` in `/usr/lib/tex/src/dvips`. You will then run `afm2tfm` over the `.afm` file to generate a `.vpl` and a raw `.tfm` file. You will run `vptovf` over the `.vpl` file to generate a `.vf` and virtual `.tfm` file. You will then install the two `.tfm` files and single `.vf` file into `/usr/lib/tex/fonts/tfm` and `/usr/lib/tex/fonts/vf`, respectively. Finally, you will add a line to the file `psfonts.map` in `/usr/lib/tex/ps` to indicate the correspondence

between the actual font name, the file the font can be found in, and the PostScript font name.

This procedure will need to be altered for some common cases. We describe a few of them with the following paragraphs.

The line that is added to `psfonts.map` will not need to contain information about where the font file can be found if the font is installed for automatic use by NeXT TEX; in this case, the generated PostScript files will not contain the relevant font program, and the NeXT PostScript spooler will automatically load the font as needed. (This is as though the font were 'resident' in the 'printer'; if the PostScript interpreter can find the font on its own, the font can be considered 'resident'.) For instance, if we installed the font `Hobo` (which has a recommended TEX name of `phbr`, and a raw name of `rphbr`) for general use on the NeXT, the corresponding entry in `psfonts.map` would simply consist of

```
rphbr Hobo
```

If, on the other hand, the font is installed for use only with NeXT TEX and not for the other NeXT applications, then the line added to `psfonts.map` must include a `<` character followed by the full path name of the font. In this case, if we put the `Hobo` font program in `/usr/lib/tex/ps`, then the appropriate line in `psfonts.map` would be

```
rphbr Hobo </usr/lib/tex/ps/Hobo
```

In this case, each time a file that used this font was printed, the font file would be included with the PostScript job; it would be the responsibility of `dvips` and `TEXview` to find and download the file, since the NeXT PostScript interpreter would not know where to find it. (This corresponds to a font not being resident in a printer.)

If you plan to export PostScript files generated by NeXT TEX to other PostScript devices, you should consider carefully whether `dvips` should include the font in the final PostScript file or not. If the font is included, then the distributed file contains a copyrighted font program; thus, the document file should not be distributed, since it might be possible to extract the font program and thus 'steal' the font. On the other hand, if the font is not included, and it is either not available on the other PostScript device, or the PostScript printing software is not smart enough to download the required font, then the file might not print. Read your font licensing agreement carefully and consider your needs.

It is often useful to have a separate configuration file for `dvips` that can be used when generating files for distribution. The standard `config.ps` file can be set up for every use on the NeXT, and a separate `config.dist` can be set up for generating distributable PostScript. This `config.dist` might contain a line like `p psfonts.dist` If `dvips` is invoked now with the `-P dist` option, then the `config.dist` configuration file would be loaded. The `p` option in this configuration file would tell `dvips` to load `psfonts.dist` instead of `psfonts.map`; this `psfonts.dist` can have the file as 'resident' or not in a different fashion than the default file.

If the font is one of the standard 35 built in to the LaserWriter, then you will not need to run `afm2tfm` or `vptovf`, nor will you need to edit `psfonts.map`, because NeXT TEX is already set up for those fonts. (Simply examine the existing `psfonts.map` from `/usr/lib/tex/ps` to determine whether a given font is one of the standard 35 or not.) The premade `.tfm` and `.vf` files for these fonts are supplied, and the `psfonts.map` file already contains entries for these fonts. You may need to edit `psfonts.map` if you do not install the fonts for general usage by other NeXT applications, so that NeXT TEX can find the font file.

If you are installing PostScript versions of standard TEX fonts, that are compatible with the standard `.tfm` widths of the fonts, then you do not want to run `afm2tfm` or `vptovf`; you simply want to add entries to `psfonts.map` after installing the fonts somewhere that either `dvips` or the NeXT PostScript interpreter can find them.

4. Case Study: Palatino-Roman

Our first case study is the installation of `Palatino-Roman` from an MS-DOS distribution of the Adobe Plus Pack (assuming that you traded in your PC for a NeXT). This font is one of the 'standard 35'. We can determine that by typing the command

```
localhost> grep Palatino-Roman /usr/lib/tex/ps/psfonts.map
```

and we notice that it is already mentioned. (Were it not, `grep` would complete without printing anything.) First, we make a directory to install the files into.

```
localhost> mkdir PlusPack ; cd PlusPack
```

Then, one by one, we take the floppy disks that the Plus Pack came on and insert them into the floppy drive and copy the contents to the current directory. The actual label of the floppy might vary (it will appear in the Workspace window); if the actual name is `unlabeled`, the following command will copy the contents:

```
localhost> cp -rp /unlabeled/* .
```

After each copy completes, eject the disk and insert the next one. (If you have problems because the disk label has a strange name, format an unlabeled DOS disk, then use a PC to copy the contents from the original disk onto the unlabeled disk.)

Once this completes, you will have a lot of files with very strange names (such as `zd_____.pfb`) in your directory. The ones with the extension `.pfb` are the font programs; the ones with the extension `.afm` are the `.afm` files; the others are not needed.

We want to install the Palatino-Roman font; which one of the files contains that font? In this case, a file called `install.cfg`. This file just happens to contain a list with the correspondence between the short disk names and the longer PostScript font names. There

should be some such list somewhere with the fonts; if there is not, you can examine the tops of the `.pfb` files, carefully, looking for the name of the font contained in that file. It should be on the first line; you might have to run `undos` over the file to turn it into ASCII before you can do this. In this case, the command

```
localhost> grep Palatino-Roman install.cfg
```

tells us that the `.pfb` and `.afm` files we want have the name `POR____`. Since the NeXT translates uppercase DOS file names to lowercase, we actually want `por____`.

Since this font is a binary file (from the `.pfb` extension), we need to run `undos` over it. We also need to create an appropriate `Palatino-Roman` font directory. Note that it is very important that we spell `Palatino-Roman` correctly in all of the commands we type; even the case is important.

First, we shall install it for general use by the NeXT. We type the commands

```
localhost> cp /usr/lib/tex/src/misc/undos .
localhost> mkdir Palatino-Roman.font
localhost> undos <por_____.pfb >Palatino-Roman.font/Palatino-Roman
localhost> cp por_____.afm Palatino-Roman.afm
localhost> cp Palatino-Roman.afm Palatino-Roman.font
```

Note that we kept a copy of `Palatino-Roman.afm` in the current directory. This was not necessary; it would only be necessary if we wanted to run `afm2tfm` to create a new `.tfm` or `.vpl` file. Since this is one of the standard 35 fonts, that will not be necessary; NeXT TEX is already configured to handle them.

Now, let's install the font in the appropriate directory. The font must be installed in one of `/NextLibrary/Fonts`, `/LocalLibrary/Fonts`, or `~/Library/Fonts`. We will make it available to everyone on the machine by installing it in `/LocalLibrary/Fonts`. First, we must become superuser, and then we install the directory.

```
localhost> su
Password:
localhost# cp -rp Palatino-Roman.font /LocalLibrary/Fonts
localhost# cd /LocalLibrary/Fonts
localhost# rm -f .afmcache .fontlist .fontdirectory
```

Now, we type control-D to exit superuser status. The font should be available for general NeXT applications, and we should be able to use it in NeXT TEX. First, we determine what it is called under NeXT TEX; we do this with

```
localhost> grep Palatino-Roman /usr/lib/tex/src/dvips/adobe
```

and we find that the corresponding (virtual, not raw) name is `pplr`. So, we test it by printing a simple font table:

```
localhost> tex testfont
This is CTeX, NeXT Version 3.141
(/usr/lib/tex/inputs/testfont.tex
Name of the font to test =
```

We now type `pplr`, and later, when requested, type `\sample\bye`, and then print with `dvips`:

```
Name of the font to test = pplr
Now type a test command (\help for help):)
*\sample\bye
[1] [2]
Output written on testfont.dvi (2 pages, 11776 bytes).
Transcript written on testfont.log.
localhost> dvips testfont
This is dvips 5.489 Copyright 1986, 1992 Radical Eye Software
' TeX output 1992.07.05:2309' -> !lpr
<tex.pro><texps.pro>.  [1] [2]
```

Out of the printer should come a nice table of the font with a paragraph testing some of the ligatures, kerns, and accents. Congratulations; you can now use the font!

As you can see, installing them all can be a royal pain, mostly because of the process of converting a MS-DOS style font into a NeXT font directory and putting it in the correct place. But it is possible. Indeed, starting with `install.cfg`, it is not difficult to put together a batch file that does almost all of the work for you.

5. Case Study: CooperBlack

Now let's go through the same process, only with `CooperBlack`, which is not a font in the standard 35. We assume that we begin with the `CooperBlack.afm` and `CooperBlack` file (already in ASCII format). We follow essentially the same process, only we also add in steps to run `afm2tfm` and `vptovf`, edit the `psfonts.map` file, and install the `.tfm` and `.vf` files.

```
localhost> mkdir temp
localhost> cp CooperBlack CooperBlack.afm temp
localhost> cd temp
localhost> mkdir CooperBlack.font
localhost> cp CooperBlack CooperBlack.afm CooperBlack.font
localhost> grep CooperBlack /usr/lib/tex/src/dvips/adobe
CooperBlack pcbr
CooperBlack-Italic pcbri
localhost> afm2tfm CooperBlack.afm -v pcbr.vpl rpcbr.tfm
rpcbr CooperBlack
localhost> vptovf pcbr.vpl pcbr.vf pcbr.tfm
This is VPtoVF, C Version 1.3
'015 '016 '017 '020 '022 '023 '024 '025
.  .  .
'326 '327 '330 '343 '350 '353 '370.
I had to round some heights by 16.5000000 units.
I had to round some depths by 3.0000000 units.
localhost> su
Password:
localhost# ex /usr/lib/tex/ps/psfonts.map
"/usr/lib/tex/ps/psfonts.map" 85 lines, 3088 characters
:$a
pcbr CooperBlack
.
:w
"/usr/lib/tex/ps/psfonts.map" 86 lines, 3105 characters
:q
localhost# cp pcbr.tfm rpcbr.tfm /usr/lib/tex/fonts/tfm
localhost# cp pcbr.vf /usr/lib/tex/fonts/vf
localhost# cp -rp CooperBlack.font /LocalLibrary/Fonts
localhost# cd /LocalLibrary/Fonts
localhost# rm -f .afmcache .fontlist .fontdirectory
```

Note how we determined the appropriate TEX name from the `adobe` file. If we did not want to install this font for general NeXT use, we would have omitted the steps creating and copying the `CooperBlack.font` file, and instead done something like (immediately after copying the `.vf` and `.tfm` files):

```
localhost# cp CooperBlack /usr/lib/tex/ps
```

In addition, the line we added to `psfonts.map` would have looked like

```
pcbr CooperBlack </usr/lib/tex/ps/CooperBlack
```

Finally, if the font was in binary `.pfb` format, we would have had to run `undos` over it.

6. Case Study: Blue Sky Research CM Fonts

Our final example is to convert and use the Blue Sky Research Computer Modern PostScript fonts. These are commercial fonts available from Blue Sky Research; their phone number is (503) 222–9571. These fonts work very well, especially when you are creating EPSF files from TEX (of, for instance, equations) for pasting into other applications; the resulting files scale much better than normal METAFONT bitmapped fonts. The only problem is that the resultant files only print correctly at sites that have the Computer Modern PostScript fonts.

In this case, we want to install the fonts so they can be used by other applications on the NeXT. Even though other applications will not normally want to use them, making them loadable in this fashion will make their operation with TEX EPSF files much faster, and it will make the EPSF files much smaller.

We also want to add entries to `psfonts.map`. But since there are so many additional entries, and since we may not always want to make use of the PostScript versions of the fonts, we will place the mappings in a separate `psfonts` file and modify `config.ps` to load this mapping file in addition.

Our process shall start with the two-disk set available for MS-DOS platforms. Just as with the Plus Pack example above, we copy the full contents of the floppies into a new directory. Follow the procedure given above.

Because there are so many fonts, we will install them with a 'batch' shell command. This shell command is supplied as `/usr/lib/tex/src/misc/fixcm`.

We have the following steps. First, we must copy the `undos` and `fixcm` executables into our local directory. We assume that you have already made a temporary directory, changed the current directory to the new one, and copied the files in.

```
localhost> cp /usr/lib/tex/src/misc/fixcm .
localhost> cp /usr/lib/tex/src/misc/undos .
```

Then, the `fixcm` program runs a loop over the set of fonts. Here is the `fixcm` program itself:

```
#!/bin/csh
rm -f psfonts.cm
foreach i (*.pfb)
set n=‘basename $i .pfb‘
set u=‘basename $i .pfb | tr a-z A-Z‘
mkdir $u.font
cp $n.afm $u.font/$u.afm
undos <$n.pfb >$u.font/$u
echo $n $u >>psfonts.cm
end
```

This shell script first removes any `psfonts.cm` file that might already exist. Then, for each `.pfb` file it finds in the current directory, it first sets the shell variable `n` to be the base name of the font, without the extension. Next, using `tr`, it coverts the name to uppercase and stores the result in the shell variable `u`; this is necessary because the PostScript font names of these fonts are full uppercase where the normal TEX name is lower case.

Then, it makes the appropriate `.font` directory, and copies the `.afm` file into it. It then runs `undos` over the `.pfb` file and stores the result under the appropriate name in the `.font` directory. Then, it adds a line to `psfonts.cm` indicating the appropriate name mapping.

This shell script can be invoked by simply typing `fixcm`; you should not be superuser when executing it. After `fixcm` has completed, type the following commands:

```
localhost> su
Password:
localhost# cp -rp *.font /LocalLibrary/Fonts
localhost# cp psfonts.cm /usr/lib/tex/ps
localhost# cd /LocalLibrary/Fonts
localhost# rm -f .afmcache .fontlist .fontdirectory
localhost# cd /usr/lib/tex/ps
localhost# ex config.ps
"config.ps" 66 lines, 1141 characters
:$a
p+ psfonts.cm
.
:w
"config.ps" 67 lines, 1155 characters
:q
```

The addition to `config.ps` says to add the contents of `psfonts.cm` to the list of PostScript fonts that are available; the `+` character means to add the list, rather than replacing the current list.

Now, get out of superuser mode. If TEXview is running, quit it. Test the results by running `Edit`, typing the equation `$$-b\pm\sqrt{b^2-4ac}\over 2a$$` into a window, selecting it, and selecting the menu entry `Service/TeXview/TeX EQ->EPS`. If what appears is recognizably the quadratic equation, congratulations! Installation worked. If what appears has large gaps in the characters, major portions of some characters missing, or dropped letters or generally is unrecognizable, well, you're still using bitmapped fonts. When printed, the two results should be the same.

A Additional Reading

This manual does not intend in any way to be an exhaustive introduction to the world of TEX and METAFONT. The following books provide a much more complete reference to the world of TEX. Most of the following books were also typeset with TEX, so they provide an excellent example of what is possible.

The TEXbook by Donald E. Knuth, Addison-Wesley, 1984. Also published as Volume A of *Computers and Typesetting*, this is the major manual to the TEX system and is a must-have for anyone using TEX. It is both a tutorial and a reference manual, and even contains a few jokes. ISBN 0-201-13448-9

LaTEX: A Document Preparation System by Leslie Lamport, Addison-Wesley, 1986. This book describes use of the LaTEX and SliTEX macro packages, and is a less painful introduction to the world of TEX. ISBN 0-201-15790-X

The METAFONT*book* by Donald E. Knuth, Addison-Wesley, 1986. Also published as Volume C of *Computers and Typesetting*, this is the major manual to the METAFONT system. ISBN 0-201-13444-6

Computers and Typesetting, Volume B: TEX: The Program by Donald E. Knuth, Addison-Wesley, 1986. This volume *is* TEX. It is a full listing of the original WEB source code, complete with beautiful formatting, an index, table of contents, and introduction. For those who really want to know how TEX works. ISBN 0-201-13437-3

Computers and Typesetting, Volume D: METAFONT*: The Program* by Donald E. Knuth, Addison-Wesley, 1986. This volume is analogous to Volume B, only for METAFONT. ISBN 0-201-13438-1

Computers and Typesetting, Volume E: Computer Modern Typefaces by Donald E. Knuth, Addison-Wesley, 1986. This volume describes those beautiful computer modern typefaces in excruciating detail. For those who intend to design their own typefaces, this book is absolutely necessary.

The Joy of TEX by M. D. Spivak, Ph.D., Addison-Wesley, 1986. This volume is an excellent introduction to TEX, using the AMSTEX macro package. ISBN 0-8218-2999-8

Graphic Design for the Electronic Age by Jan V. White, Watson-Guptill Publications, 1988. This book is on graphic design in general, and makes no references to TEX; nonetheless, it is full of good, practical information on typography and page design.

TEX for the Impatient by Paul W. Abrahams with Karl Berry and Kathryn A. Hargreaves, Addison-Wesley, 1990. An excellent beginners book on TEX, this book is laid out to be useful to the seasoned user of TEX, as well. ISBN 0-201-51375-7

LaTEX for Scientists and Engineers by David J. Buerger, McGraw-Hill, 1990. Intended to be an introductory text with examples for professionals who must cope with complex equations, this book supplies much information that was lost or muddled in the LaTEX book.

A Permuted Index for TEX and LaTEX by Bill Cheswick, AT&T Computing Science Technical Report, . This document, free for the asking, lists all TEX and LaTEX commands and permutes the listing so you can easily find the ones you need. The request address is:

Comp. Sci. Tech. Reports
AT&T Bell Laboratories
Room 2C-579
600 Mountain Avenue
Murray Hill, NJ 07974
(908) 582-2918
Courtney Kmosko

PostScript Language Tutorial And Cookbook by Adobe Systems Incorporated, Addison-Wesley, 1986. An introduction to the PostScript language for users. If you have a PostScript printer, this and the next book should be right next to it. Using raw PostScript is easier than you'd think. ISBN 0-201-10189-0

PostScript Language Reference Manual by Adobe Systems Incorporated, Addison-Wesley, 1990. This is the main reference defining the PostScript language. If you are serious about using PostScript, it is a must-have. ISBN 0-201-18127-4

TEX by Topic, A TEXnician's Reference by Victor Eijkhout, Addison-Wesley, 1992. This book provides a more in-depth description of TEX, and presents much detailed information in a different organization than the TEXbook. I guarantee you will learn something when you read this book. ISBN 0-201-56882-9

TEX by Example, A Beginner's Guide by Arvind Borde, Harcourt Brace Jovanovich, Publishers, 1992. This book provides excellent examples of TEX side by side with the actual TEX code used to create that page, in fully commented detail. It is a joy to read. ISBN 0-12-117650-9

A Beginner's Book of TEX by R. Seroul and S. Levy, Springer-Verlag, . This book is a fairly streamlined introduction to plain TEX that also contains lots of little hints and tricks

for moderate to advanced users. The index doubles as a glossary/quick reference to plain TEX commands. ISBN 0-387-97562-4

B Building on NeXTTEX

NeXTTEX is a complete package in itself, with a myriad of options and capabilities. But many additional packages are available that can extend the capabilities or ease of use of NeXTTEX. This appendix lists just a few of these.

1. AUC TEX

AUC TEX is an Emacs lisp mode for editing, processing, previewing, and sending to `dvips` of TEX and LaTEX files. It was written by Kresten Krab Thorup, Dept. of Math and Computer Science, University of Aalborg, Aalborg, Denmark, `krab@iesd.auc.dk` and is maintained by `auc-tex_mgr@iesd.auc.dk`.

It is available for ftp from `iesd.auc.dk` and several places in the United States. AUC TEX consists of a series of emacs-lisp files, which allow one to run plain TEX or LaTEX directly from Gnu-emacs, with command completion, and a large number of shortcuts.

AUC-TeX is available by anonymous ftp to `iesd.auc.dk` as `/pub/emacs-lisp/auc-tex-X.x.tar.Z`, (where `X.x` specifies the release version). Also, it should be available at major TEX and Elisp archives around the world. In case you don't have access to anonymous ftp, you can get it by electronic mail requests to `auc-tex_mgr@iesd.auc.dk`.

AUC TEX automatically indents your source. It has a special outline feature, which can greatly help you 'getting the overview' of a document.

Apart from these special features, AUC TEX provides an large range of handy Emacs macros, which in several different ways can help you write your LaTeX documents fast and painless.

All features of AUC TEX are documented using the GNU Emacs online documentation system. That is, documentation for any command is just a key click away!

AUC TEX is written entirely in Emacs-Lisp, and hence you can easily add new features for your own needs. It was not made as part of any particular employment or project (apart from the AUC TEX project itself). AUC TEX is distributed under the 'GNU Emacs General Public License' and may therefore almost freely be copied and redistributed.

2. Blue Sky Research PostScript Fonts

Blue Sky Research (534 Southwest Third Avenue, Portland, Oregon) sells PostScript type-1 versions many METAFONT fonts. Use of these fonts eliminates the piles of `pk` fonts that clutter up your machine (although they do take up a fair amount of space themselves.) They also allow TEX to generate resolution-independent clip-art. Since generating fonts by PostScript font scaling is often much faster than generating fonts by METAFONT, new sizes of the fonts can be available much more rapidly, and previewing in TEXview is thus initially sped up.

3. InstantTEX

This program instantly previews your files as you edit the TEX source. A few seconds after you type a key, the new output appears in TEXview. This is very useful for writing equations, macros, adjusting the spaces and sizes in your file, and almost everything else that you can do with TEX. This feature works well even on multiple-page documents because of the speed of the NeXT.

If TEX notices an error in your file, you can press a key and InstantTEX will go to the line which contains the error. This considerably simplifies editing your text.

You can TEX your file in any format you wish, directly from Edit, just by pressing a key. The format of you TEX files can be determined automatically.

There is an online reference card which helps you write Greek letters, operators, relations, and other TEX symbols. There are more than 20 various options to suit your preferences.

InstantTEX is available from Instant Technology, 744 Mayfield Avenue, Stanford CA 94305. You can send electronic mail to `dmitri@StarConn.com`. A demo version of InstantTEX and the latest information is available on the public archive `nova.cc.purdue.edu`.

4. TEXmenu

With TEXmenu you can start `plain` TEX, LaTEX, SliTEX, AmsTEX or TEX with any custom format, edit your TEX files with Edit, start BibTEX, MakeIndex and make, manage your TEX projects and files, automatically determine the format of you TEX files, and check spelling, all with the click of a button.

The author of TEXmenu, Harald Schlangmann, made it available for free on the public archive `nova.cc.purdue.edu`.

F Font Magnification

This appendix contains some basic information on font magnification with the supplied Computer Modern fonts.

TEX defines seven standard magnifications; these are called magsteps. The macros and corresponding magnification ratios are:

`\magstep0`	1.000
`\magstephalf`	1.095
`\magstep1`	1.200
`\magstep2`	1.440
`\magstep3`	1.728
`\magstep4`	2.074
`\magstep5`	2.488

You will note that magstep n is equivalent to a magnification of 1.2^n. To make a document use a specific magnification for each of its fonts and other dimensions, simply place a command such as

```
\magnification=\magstep1
```

Try it on one of your documents, and see the difference. All dimensions specified without a `true` prefix (as in the standard `baselineskip`, for instance) will be magnified; those specified in `true` sizes, such as `hsize` and `vsize`, will not be changed.

If you just need a larger font for headers or something, you can define such a font with a command like

```
\font\bigbold=cmbx10 scaled \magstep1
```

Now you can use the `bigbold` font just as you might use the `it` font. If you use a scaled font in a document that also has a magnification statement, you must multiply the two factors (or add the magsteps) to determine which font will actually be used. For instance, if you have a document with magnification equal to magstep half, and you load a font scaled magstep half, the actual font will be loaded at a size of magstep 1. But if you try to load

a font scaled magstep 1, the font the driver will require is one at magstep 1.5, which is not supplied by default.

Such fonts can (and will) be automatically generated by `dvips`, but if completely arbitrary magnifications are used, the font cache will soon become cluttered with many fonts that are never used. In addition, other TEX systems without the automatic font generation capability will often not have these fonts, so your documents will lose their portability. When at all possible, try to use standard magstep values.

The scaled parameter (and the magnification parameter) is in thousandths. Thus, if you type to TEX,

```
\message{\magstep1}
```

TEX will display '`1200\relax`'. (You can type to TEX '`\show\magstep`' to understand why.) So, if I want to use a magstep 2 font (that's a magnification of 1.44) on a document magnified at magstep half (that's 1.095), I would scale it at 1000*(1.44/1.095), or 1315, as in:

```
\font\bigbold=cmbx10 scaled 1315
```

There are also different point sizes of the same font. For instance, the standard roman font `cmr10` is supplied in point sizes of 5, 6, 7, 8, 9, 10, 12, and 17 points. It is usually better to use these point sizes than magnified versions of ten points, because the fonts are designed for unmagnified use. But most people probably won't be able to tell the difference.

There are some cases, such as logos and the like, where you want to write a macro which refers to a font at a specific size, independent of the magnification of the document which encloses it. In this case, you can define a font as in

```
\font\bigbold=cmbx10 at 14.4 truept
```

This will define `cmbx10` at a magnification of magstep 2, no matter what the magnification of the document.

How does all of this relate to the names of the `pk` files? Actually, it's quite simple. The font `cmr12`, for instance, for a 180 dot per inch dot matrix printer, at magstep 3, is named `cmr12.311pk`. The `311` can be calculated by multiplying 180 by 1.2^3, and rounding to the nearest integer.

G Glossary

The TEX world has its own set of jargon, acronyms and technical terms. The NeXT and Unix world is not free from them either. This section contains the definitions of some of the more obscure terms used in this manual.

`.afm` file This is an ASCII file containing information on the sizes of characters, ligatures, kerns, and the font encoding of a particular PostScript font. It is used by `afm2tfm` to generate the files TEX needs to use PostScript fonts.

desktop metaphor A paradigm of user interface where files are represented as icons, programs as windows, and the user 'executes' commands by selecting, dragging, and pointing with the mouse. For instance, to delete a file using the desktop metaphor, you would click on the icon you want to delete, and drag the icon over to the black hole or trashcan, and let go of the select button.

dither To turn a color or multi-level picture into a bilevel picture, or to reduce the number of color or grey levels in a picture. There are a large number of ways to do this; `pfilt` provides some of the most effective.

double click Two rapid presses of the select button; this is the way an icon is activated in the NeXT machine. If the icon represents a program, the program is run; if the icon represents a directory, the directory is opened and a window showing the contents of the directory is displayed.

driver A program that takes a TEX device-independent (or `.dvi`) file and interprets it, loading the rasters for the fonts as necessary, and sends the required commands to a particular printer or other output device to produce the document described.

`.dvi` file A device-independent file is the output of the TEX program; it contains a description of the typeset document in a compact form that is resolution and output device independent.

environment variables Global variables that the user can set and which are accessible to programs in the machine. An environment variable can be used to set some option or convey some information to a program; for instance, they might be used to tell TEX where to find font files in a user-configured system.

`.fmt` file A format file is a compact form of a set of TEX macros that is loaded in every time TEX is run to make those macros available. The `fmt` file is created by iniTEX and consists of essentially a memory image of TEX after the macros have been loaded. The default `fmt` file is called plain TEX; LaTEX uses another format file called `lplain.fmt`. The format files must reside in the `/usr/lib/tex/formats` directory.

font A collection of printing characters of one style or size; for instance, `cmr10` is Computer Modern Roman at 10 points. The word font is used to refer to both a particular style, independent of size, and a particular style at each individual size. NeXTTEX is currently supplied with the standard 102 TEX fonts; many different sizes of these fonts are distributed for the previewer.

`.gf` file A generic file containing raster information for a particular font at a particular resolution; the same information in the packed or `.pk` font files.

icon Some graphics imagery that represents a file, program, device, or directory. These can be activated or opened by double-clicking on them with the mouse.

iniTEX The iniTEX program is a larger version of TEX that takes more memory to run; its purpose is to create format files from a set of macros.

kerning The addition or removal of spaces between characters due to their shape. For instance, in the word 'AVAILABLE', the 'A' and 'V' are moved closer together; compare this with 'AVAILABLE', with the kerning turned off. TEX handles kerning automatically.

LaTEX A macro package intended to both make TEX easier to use and more powerful, LaTEX was written by Leslie Lamport.

ligature A ligature is the combination of two or more characters into one character. For instance, the 'f' and 'i' in 'file' is a ligature; compare this with 'file', using separate characters instead. TEX handles ligatures automatically; no special characters need be typed.

`.log` file A file that contains all of the terminal output during a run of TEX or METAFONT; it is created automatically along with the device-independent file or font files for later perusal of any errors. It serves no purpose other than user enlightenment, so it can be safely deleted at any time.

menu button The right mouse button. This is called a menu button because pressing it displays the menus for the currently active window on the screen title bar. To select a particular menu item, you hold the menu button down, move the mouse to the menu name and then to one of the items that will appear, and then release the menu button.

`.pk` file A packed file containing raster information for a particular font at a particular resolution in a compressed format. TEX does not use these files at all, but the preview program and any other drivers require them at certain resolutions.

plain format A basic macro package that contains several hundred predefined macros; TEX loads these in at the start of each job unless another format file is specified.

pool file TEX has over 25000 characters worth of error and other messages. Rather than place these in the executable image, they are put into a special `tex.pool` file that iniTEX reads and places in every format file.

raster A graphics image stored as actual pixels rather than an analytical description. For instance, the TEX packed files contain the raster, or pixel by pixel representation, of each character in a particular font. Raster graphics are by their very nature resolution dependent; analytical graphics are not.

resolution In raster graphics, the resolution refers to the number of dots (or pixels) per unit measure, usually per inch. For instance, a dot matrix printer might print at a resolution of 72 dots per inch, a laser printer at 300 dots per inch, and a typesetting machine at 1270 dots per inch.

select button The left mouse button; used to select gadgets or icons. Gadgets are selected by moving the pointer over the gadget and hitting the select button. Icons are selected the same way, but are activated by hitting the select button twice rapidly. The select button can be used to drag windows around, or to resize them, by selecting the proper gadget and holding the select button down while moving the mouse.

SliTEX A macro package written to create slides; part of the standard LaTEX distribution by Leslie Lamport.

`.sty` file An auxiliary file read in by LaTEX that describes the layout of a particular document style; `rep10.sty` describes a ten-point version of the report style. These must reside in the `/usr/lib/tex/inputs` directory when running LaTEX.

`.tfm` file A font metric file containing resolution-independent information about a particular font, such as character sizes and ligature and kerning information. TEX reads these files when you declare a new font; they must reside in the `/usr/lib/tex/fonts` directory.

`.vf` file A virtual font file contains a binary description of a font that is composed of other fonts. For instance, PostScript fonts are supported by NeXTTEX through the use of these virtual fonts, which do the character set recoding and other transformations required by TEX.

`.vpl` file This file contains an ASCII representation of the data in a `.vf` file; it can be converted to a `.vf` file with `vptovf` and back with `vftovp`.

window A window is a rectangular section of a screen that a program uses for user interaction. Several windows may all reside on one screen, and can overlap each other arbitrarily.

Index

A

a3 46
a4 46
Abrahams, Paul W. 99
accents 17, 31
Adams, Ansel 8
address 1
`adobe` 93
Adobe font metric files 88, 105
Adobe Type Manager 92
`afm` 30, 88
`afm2tfm` 6, 21, 30
`afmcache` 90
aspect ratio 4
AUC TEX 101
automatic font generation 21

B

back-tick 4
BBS 1
beautiful 11
Berry, Karl 99
betrothed 11
bibliography 98
BIX 1
Blue Sky Research 79, 96, 102
`bop-hook` 24, 56, 80
Borde, Arvind 99
bounding box 23
Bowie, David 4
break 12
Buerger, David J. 99
bugs 8
`buildafmdir` 90

C

changes 4
Cheswick, Bill 99
click dragging 71
clipping 4
color 4
command keys 71
`commandposition` 80
`compressed` 80
compression 48
Computer Modern 103
`config.ps` 48, 49, 91
console 6, 73
`consoleposition` 80
control-D 47
CooperBlack 94
copies 44, 47
Coursey, Janet 11
crop marks 5
`customdpi` 80
`customformat` 80

D

dancing 11
debug 45, 81
`defaultfont` 81
`defaultformat` 81
desktop 105
directory 70
dither 105
dvi file 10, 70, 105
`dvips` 20, 77
`DVIPSHEADERS` 55
`dvipsrc` 20

`dvitype` 84

E

Edit 17
editor 9, 17
Eijkhout, Victor 99
emergency exit 12
`emtex` 21
Encapsulated PostScript 5
encoding 5, 17
`end-hook` 56, 80
environment variables 7, 16, 70, 105
EOF 47
`eop-hook` 56, 80
`epsf` macros 24
`epsfsize` 25
`epsfxsize` 25
exit 12
exitserver 89
expanded fonts 36
extension 15

F

FAX 2, 7
filter 45, 51
`fixcm` 96
`fli` 58
font metric file 10, 107
`fontdirectory` 90
`fontlist` 90
`foo.tex` 9
format file 4, 10, 12, 13, 16, 73, 106, 107

G

`generatefonts` 81
generic font file 106
`gftodvi` 84
`gftopk` 85
`gftype` 86
glossary 105
graphics 4, 23
gray fonts 84

H

Hargreaves, Kathryn A. 99
header 26, 45, 51, 52
help 5, 6
`hmarg` 81
`HOME` 54, 55, 82
`hsize` 81

I

icon 106
`initex` 15, 17
installation 2, 58, 90
`Installer` 2
InstantTEX 102
international character sets 17
interrupt 12

K

kerning 32, 106
keyboard shortcuts 76
Knuth, Donald E. 9, 30, 98

L

Lamport, Leslie 10, 98
landscape 8, 22, 46
`leavevmode` 24
ledger 46
legal 46
letter 46
Levy, S. 99
ligature 6, 31, 32, 106
Linde, Dmitri 11, 102
links 12
literal PostScript 27
log file 10, 106
Loyola, John 11

M

magnification 24, 46, 103
`magscale` 25

magstep 103
MAKETEXPK 4, 52, 54, 55, 59, 81, 82
manual feed 45
Marchioro, Thomas 11
maxdrift 45, 51
measurement 7
memory 5, 20, 32, 51
menu button 106
METAFONT 10, 49, 52
METAPOST 21
mf.pool 107
mft 86
misfeatures 8
mocked 6, 81
mode 46
mode_def 49
MS-DOS 21, 37, 57

N

NeededFonts 75
new font selection scheme 17
next.tex 17
NextCD/Packages 2
NFSS 17

O

obsolete 27
open 70
output 46, 51

P

packed font file 10, 107
pages 7, 45, 46
Palatino-Roman 92
paper size 5
paper type 5, 8, 46
paranoia 81
patgen 86
path list substitutions 8
pfa 88
pfb 5, 36, 88
pfm 88
pk 52, 53, 56, 82
pktogf 86
pktype 86
plain 12
plain.base 49
pltotf 87
portrait 22
posters 6
PostScript 20
PostScript fonts 8, 30, 80, 88
PostScript graphics 23, 80
Preferences 7, 74
PRINTER 55
printing 3, 20, 77
problems 8
proof sheets 84
psfig 21
psfile 28
psfonts.map 5, 32, 35–37, 91
psfonts.sty 8

Q

quiet 46, 52

R

Radical Eye Software 1, 84
resolution 21, 47, 48, 52, 53, 77, 80–82
reverse 46, 52, 81
reversed 81
rulers 7

S

scaleunit 28
scaling 4
Schlangmann, Harald 102
SDict 27, 28
sectioning 5
select button 107
Seroul, R. 99
services 6, 78
setenv 16
sexy 11
Skulina, Susan 11
slanted fonts 36

Spivak, Michael 98
Stanford University 9
`start-hook` 56, 80
stop 12
structured comments 47
style file 107
support 1
`sync` 81

T

TEXmenu 102
`tex.pool` 17, 107
`tex.remap` 4, 18
`TEXCONFIG` 56, 82
`TEXEDIT` 17
`TEXFONTS` 16, 53, 56, 82
`TEXFORMATS` 16
`TEXINPUTS` 16, 53, 56, 82
`TEXPKS` 52, 56, 82
`TEXPOOL` 17
`texposition` 81
`TeXview.service` 78, 79
`TeXview_Pipe` 14
`tfm` 16, 30, 37, 56, 82, 92
`tftopl` 87
Thorup, Kresten Krab 101
`tpic` 21

U

`undos` 89, 96
UNIX 21, 54
Unix commands 14
unzoom 76
`unzoomres` 81

V

`vf` 53, 90, 92
`VFFONTS` 56, 83
`vftovp` 87
virtual fonts 31, 53, 90, 107
virtual memory, PostScript 6
virtual property list file 107
`vmarg` 81
VMS 21
`vpl` 90
`vptovf` 87
`vsize` 81

W

White, Jan 99
wife 11
window 108
`windowposition` 82
windows 71
Workspace 2, 3, 70
WYSIWYG 9

Z

zoom 76
`zoomres` 82

NeXTTEX

An Implementation of TEX for the NeXT Computer

Version 3.141
1 August 1992

$$\sqrt{i}$$

This document describes the NeXTTEX software.

Manual Copyright

Disclaimer

Radical Eye Software makes no representation or warranty with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Radical Eye Software reserves the right to revise this document and the enclosed software without obligation of Radical Eye Software to notify any person of such revision.

Trademarks

TEX is a trademark of The American Mathematical Society. NeXT is a trademark of NeXT Incorporated. Unix is a trademark of Bell Laboratories. IBM is a trademark of International Business Machines. HP and LaserJet are trademarks of Hewlett-Packard Incorporated. PostScript is a trademark of Adobe Incorporated. Dr. Pepper is a trademark of the Dr. Pepper Company. Linotronic is a trademark of the Linotype Company.

Table of Contents

Chapter 1: Introduction **1**
1. First Run 2
    1.1 Installing TEX 2
    1.2 Running TEX 3
    1.3 Printing a Document 3
2. What Is New for 3.0? 4
    2.1 Changes to TEX 4
    2.2 Changes Shared Between TEXview and Dvips 4
    2.3 Changes to Dvips 5
    2.4 Changes to TEXview 6
    2.5 Incompatible Changes 8
    2.6 Other Changes 8
    2.7 Known Problems 8
3. What is TEX? 9
4. Acknowledgements 11

Chapter 2: Using TEX **12**
1. Exiting TEX 12
2. TEX Format Files 12
    2.1 Specifying a Format File on the Command Line 13
    2.2 Specifying a Format File in a TEX File 13
3. Communicating with TeXview 14
4. Invoking Unix Commands from TEX 14
5. IniTEX 15
6. NeXT Environment Variables 16
7. The New Font Selection Scheme 17
8. International Character Sets 17
    8.1 Poor Man's Extended Characters 17
    8.2 Extended Characters Done the Right Way 18

Chapter 3: Printing TEX Documents **20**
1. Why Use dvips? 20
2. Using dvips 21
3. Paper Size and Landscape Mode 22
4. Including PostScript Graphics 23
    4.1 The Bounding Box Comment 23
    4.2 Using the EPSF Macros 24
    4.3 Header Files 26
    4.4 Literal PostScript 27
    4.5 Literal Headers 27
    4.6 Other Graphics Support 27
    4.7 Dynamic Creation of PostScript Graphics Files 30
5. Using PostScript Fonts 30
    5.1 The afm2tfm Program 31
    5.2 Changing a Font's Encoding 33
    5.3 Special Effects 35

5.4 Non-Resident PostScript Fonts . . . 36
5.5 Font Aliases . . . 37
6. Font Naming Conventions . . . 38
6.1 Foundry . . . 40
6.2 Typeface Families . . . 40
6.3 Weight . . . 40
6.4 Variants . . . 41
6.5 Expansion . . . 41
6.6 Naming Virtual Fonts . . . 42
6.7 Examples . . . 42
7. Command Line Options . . . 44
8. Configuration File Searching . . . 49
9. Configuration File Options . . . 49
10. Automatic Font Generation . . . 54
11. Path Interpretation . . . 54
12. Environment Variables . . . 55
13. Other Bells And Whistles . . . 56
14. MS-DOS . . . 57
15. Installation . . . 58
16. Diagnosing Problems . . . 60
16.1 Debug Options . . . 60
16.2 No Output At All . . . 60
16.3 Output Too Small or Inverted . . . 61
16.4 Error Messages From Printer . . . 61
16.5 400 DPI Is Used Instead Of 300 DPI . . . 61
16.6 Long Documents Fail To Print . . . 61
16.7 Including Graphics Fails . . . 62
16.8 Can't Find Font Files . . . 62
16.9 Can't Generate Fonts . . . 62
17. Using Color with dvips . . . 63
17.1 The Macro Files . . . 63
17.2 User Definable Colors . . . 65
17.3 Subtleties in Using Color . . . 65
17.4 Printing in Black/White, after Colorizing . . . 66
17.5 Configuring dvips for Color Devices . . . 66
17.6 Protecting Regions From Spurious Colors . . . 67
17.7 Color Support Details . . . 68

Chapter 4: Using TEXview . . . 70
1. Basic Operation . . . 70
2. Windows . . . 71
2.1 The Preview Window . . . 71
2.2 The TEX Window . . . 72
2.3 The Console Window . . . 73
2.4 The Command Window . . . 73
2.5 The Preferences Window . . . 74
2.6 Other Windows . . . 75
3. Zoom and Unzoom Resolutions . . . 75
4. Menu Options . . . 76

5. Printing from TEXview 77
6. Services 78
6.1 Predefined Services 78
6.2 User-Defined Services 79
7. PostScript Graphics and Fonts 80
8. Defaults Variables 80
9. Environment Variables 82

Chapter 5: Additional Utilities 84
1. dvitype 84
2. gftodvi 84
3. gftopk 85
4. gftype 86
5. mft 86
6. patgen 86
7. pktogf 86
8. pktype 86
9. pltotf 87
10. tftopl 87
11. vftovp 87
12. vptovf 87

Chapter 6: Using PostScript Fonts 88
1. Font Files 88
2. Installing Fonts for the NeXT 90
3. Installing Fonts for NeXT TEX 90
4. Case Study: Palatino-Roman 92
5. Case Study: CooperBlack 94
6. Case Study: Blue Sky Research CM Fonts 96

Appendix A: Additional Reading 98

Appendix B: Building on NeXTTEX 101
1. AUC TEX 101
2. Blue Sky Research PostScript Fonts 102
3. InstantTEX 102
4. TEXmenu 102

Appendix F: Font Magnification 103

Appendix G: Glossary 105

Appendix I: Index 109