Your input is needed to answer these questions. Feedback from those of you who have been actively working on porting Pascal TEX to new architectures is especially welcome. Please respond!

* * * * * * * * * * *

Software

* * * * * * * * * * *

THE FORMAT OF TEX'S DVI FILES VERSION 1

David Fuchs
TEX Project, Stanford University
April 18, 1981

When TEX compiles a document, it produces an output file that contains specifications of how TEX has decided the formatted text should appear in hard copy. These output files are known as '.DVI' files, which stands for 'device independent'. For instance, running TEX and telling it to `\input dviinf` will cause TEX to look for a file called `DVIINF.TEX`, read it, and produce an output file called `DVIINF.DVI`, which is a `.DVI` file. This document describes the format of `.DVI` files in detail, giving all the specifications along with examples.

A `.DVI` file contains information about where characters go on pages. The format is such that there are those who say that almost any reasonable device can be driven by a program that takes `.DVI` files as input. In particular, a `.DVI` file can be printed on the Xerox Dover, Xerox Graphics Printer (XGP), Varian, Versatec, Canon and Alphatype at the Stanford CS Dept., depending on what spooler it is passed to.

The `.DVI` file is a stream of 8-bit bytes, packed in computer words high-order byte first. If the computer word length is not evenly divisible by 8, then the extra bits at the low-order end of each word will be unused. The first byte in a `.DVI` file is byte number zero, the next is number one, etc. For example, on Stanford's 36-bit word machines, byte number 0 is in the highest order eight bits of the first word in a `.DVI` file, while byte number 7 is in the twelfth through fifth least significant bits of the second word in the file; and the least significant four bits in every word are zero.

A `.DVI` file is actually a series of **commands**. A **command** consists of one byte containing the **command**'s unique number, followed by a number (possibly zero) of **parameters** to the **command**. A given **command** always has the same number of **parameters**. These **parameters** may take from one to four bytes each, but a given **parameter** of a given **command** always takes the same number of bytes. Some **parameters** may sometimes be negative, in which case two's complement representation is used. The complete list of **commands**, with a description of all the `.DVI` **commands** and their **parameters**, is below. The reader is encouraged to refer to the **command** list while reading the various examples in this document.

In the **command** descriptions, a lower case letter with a [bracketed] number following it means that the **command** has a **parameter** that is that number of bytes long. An X3 **command**, for instance, is 3 bytes long, the first byte of which has the decimal value 144, the second and third of which give the distance to move to the right. If the second byte $= S$ and the third $= T$, then the distance to move is 2^8S+T (but if the high order bit of S is a one, then the distance to move is 2^8S+T-2^{16}, considering S and T as being in the range [0..255]).

The `.DVI` file contains a number of **pages** followed by a **postamble**. A **page** consists of a BOP **command**, followed by lots of other **commands** that tell where the characters on the page go, followed by an EOP **command**. Each EOP **command** is immediately followed by another BOP **command**, or by the PST **command**, which means that there are no more **pages** in the file, and the remaining bytes in the `.DVI` file are the **postamble**. Remember that TEX really doesn't have an official knowledge of **page** numbers (although it does print the value of `\count0` on your terminal as it outputs each **page** on the assumption that some meaningful number is there), so the only thing that can be said about the ordering of pages in a `.DVI` file is: The order in which **pages** come in a `.DVI` file is the same order in which TEX constructed them, which is the same order in which the TEX user specified them. Any blank or nonexistent **page** from a TEX job might not be in the `.DVI` file at all. If we consider the **page** number to be the value of `\count0`, then the **page** following **page** number 34 in a `.DVI` file might well be **page** number −5.

Some **parameters** of `.DVI` **commands** are **pointers**. A **pointer** is simply a byte number as discussed above. A **pointer** itself is 4 bytes long. For example, a BOP **command**'s last **parameter** (p[4]) is the BOP's **previous page pointer**. This parameter is the number of the byte in which the previous **page**'s BOP command begins. In particular, the *second* **page**'s BOP **command**'s **previous page pointer parameter** (p[4]) is always zero, since the first **page**'s BOP is always in byte zero in a `.DVI` file. If the first **page** in a `.DVI` file had only a BOP and EOP **command**, then the *third* **page**'s BOP's **previous page pointer**

would be 46, since the first page's BOP command takes bytes zero through 44, the first page's EOP is byte 45, so the *second* page's BOP is in byte 46.

When a .DVI-reading program reads the commands for a page, it should keep track of the current font. This can be done with a single integer variable, the value of which will always lie in the range $[0..2^{32} - 2]$. The value of the current font is changed only by FONT and FONTNUM commands. Whenever a command occurs in the .DVI file that causes a character to be set on the page, the character is implicitly from the current font.

Likewise, the program should keep track of the current position on the page. The current position on the page is like a cursor on the page; whenever a character or rule is set, it gets put at the current position on the page. The current position on the page is just two numbers—which are called horizontal coordinate and vertical coordinate. Moving to the right on a page is represented by an increase in horizontal coordinate, while moving down is an increase in vertical coordinate. The upper-left-hand corner of the page is horizontal coordinate = vertical coordinate = 0 (i.e., our system is slightly non-cartesian). Both coordinates are given in rsu's (ridiculously small units), where 1rsu = 10^{-7}meter. This is so that accumulated errors will be insignificant even in the worst imaginable case (a "box" many feet long). The current position on the page is moved about by the commands W0, W2, W3, W4, X0, X2, X3, X4, Y0, Y2, Y3, Y4, Z0, Z2, Z3 and Z4: The vertical coordinate is changed by Y and Z commands, while the horizontal coordinate is changed by W and X commands. (The value of horizontal coordinate can also change as a side effect of setting a character or rule (VERTCHAR and VERTRULE commands)—the current position on the page moves right the natural width of the character or rule set. The POP command may also change current position on the page.)

So, whoever or whatever reads a .DVI file might have three variables, F, H and V, to keep track of the current font and the current position on the page. Four more variables are also called for: w-amount, x-amount, y-amount, and z-amount. These variables hold not locations, but distances (in rsu's). The amount variables are used in .DVI files to move the current position on the page around: The commands X0 and W0 add x-amount and w-amount to horizontal coordinate, respectively, while Y0 and Z0 add y-amount or z-amount to vertical coordinate, respectively. There are also a number of commands that change the value of w-amount, x-amount, y-amount or z-amount (W2, W3, W4, X2, X3, X4, Y2, Y3, Y4, Z2, Z3 and Z4; these commands also change horizontal coordinate or vertical coordinate). Actually, the .DVI-reading program must have a stack that can hold horizontal coordinates and vertical coordinates, as well as w-, x-, y-, and z-amounts. These six values always get pushed and popped together, and a reasonable maximum stack depth might be about 200 (times six, since six items get pushed at once). As each page starts, a .DVI reading program should set the amount variables to zero. The stack should be empty. The initial value of F doesn't matter, since *every* page of a .DVI file must have a FONT or FONTNUM command before any command that will set a character (the HORZCHAR and VERTCHAR commands). Note that F is *not* pushed and popped.

A program called DVITYP is available that takes any .DVI file and prints a readable description of its contents, together with error messages if the file is not in the correct format.

. . . . . .

Command Name
: Command Bytes
: Description

VERTCHAR0
: 0
: Set character number 0 from the current font such that its reference point is at the current position on the page, and then increment horizontal coordinate by the character's width.

VERTCHAR1
: 1
: Set character number 1, etc.

⋮ ⋮

VERTCHAR127
: 127
: Set character number 127, etc.

NOP 128
: No-op, do nothing, ignore. Note that NOPs come *between* commands, they may not come between a command and its parameters, or between two parameters.

BOP 129 c0[4] c1[4] ... c9[4] p[4]
Beginning of page. The parameter p is a pointer to the BOP command of the *previous* page in the .DVI file (where the *first* BOP in a .DVI file has a p of −1, by convention). The ten c's hold the values of TEX's ten \counters at the time this page was output.

EOP 130
The end of all commands for the page has been reached. The number of PUSH commands on this page should equal the number of POPs.

PUSH 132
Push the current values of horisontal coordinate and vertical coordinate, and the current w-, x-, y-, and z-amounts onto the stack, but don't alter them (so an X0 after a PUSH will get to the same spot that it would have had it had been given just before the PUSH).

POP 133
Pop the z-, y-, x-, and w-amounts, and vertical coordinate and horisontal coordinate off the stack. At no point in a .DVI file will there have been more POPs than PUSHes.

HORZRULE
135 h[4] w[4]
Typeset a rule of height h and width w, with its bottom left corner at the current position on the page. If either $h \leq 0$ or $w \leq 0$, no rule should be set.

VERTRULE
134 h[4] w[4]
Same as HORZRULE, but also increment horisontal coordinate by w when done (even if $h \leq 0$ or $w \leq 0$).

HORZCHAR
136 c[1]
Set character c just as if we'd gotten the VERTCHARc command, but don't change the current position on the page. Note that c must be in the range [0..127].

FONT 137 f[4]
Set current font to f. Note that this command is not currently used by TEX—it is only needed if f is greater than 63, because of the FONTNUM commands below. Large font numbers are intended for use with oriental alphabets and for (possibly large) illustrations that are to appear in a document; the maximum legal number is $2^{32} - 2$.

X2 144 m[2]
Move right m rsu's by adding m to horisontal coordinate, and put m into x-amount. Note that m is in 2's complement, so this could actually be a move to the left.

X3 143 m[3]
Same as X2 (but has a 3 byte long m parameter).

X4 142 m[4]
Same as X2 (but has a 4 byte long m parameter).

X0 145
Move right x-amount (which can be negative, etc).

W2 140 m[2]
The same as the X2 command (i.e., alters horisontal coordinate), but alter w-amount rather than x-amount, so that doing a W0 command can have different results than doing an X0 command.

W3 139 m[3]
As above.

W4 138 m[4]
As above.

W0 141
Move right w-amount.

Y2 148 n[2]
Same idea, but now it's "down" rather than "right", so vertical coordinate changes, as does y-amount.

Y3 147 n[3]
As above.

Y4 146 n[4]
As above.

Y0 149
Guess.

Z2 152 m[2]
Another downer. Affects vertical coordinate and z-amount.

Z3 151 m[3]

Z4 150 m[4]

Z0 153
Guess again.

FONTNUM0
154
Set **current font** to 0.

FONTNUM1
155
Set **current font** to 1.

⋮ ⋮

FONTNUM63
217
Set **current font** to 63.

PST 131 p[4] n[4] d[4] m[4] h[4] w[4]
Fontdef Fontdef ... Fontdef
-1[4] q[4] i[1] 223[?]
The **postamble** starts here. See below for the full explanation of the **parameters** of the **postamble**.

Commands 218–255 are currently undefined and will not be output by TEX.

The PST **command**, which is always the last command in a .DVI file, is somewhat special. The **parameter** p is a **pointer** to the BOP of the final **page** in the .DVI file. The **parameters** n and d are the numerator and denominator of a fraction by which all the dimensions in the .DVI file should be multiplied by to get rsu's (TEX always outputs a 1 for each of these values, they are included in .DVI format to allow other text systems to conveniently output .DVI files). The **parameter** m is the overall magnification requested by par12 in the TEX job (par12 is unitless, and is 1000 times the desired magnification). Next come h and w, which are the height of the tallest **page**, and the width of the widest (both in rsu's).

Next in the postamble come the **font definitions**, one for each font used in the job (i.e., each FONT and FONTNUM **command** in a .DVI file must refer to a **font number** that has a **font definition**). The format of a **font definition** can be considered to be:

fnum[4] fchk[4] fmag[4] fnamlen[1] fnam[fnamlen]

The **font number** is held in fnum. The **font checksum** (from the font's TFM file) is in fchk. The **parameter** fmag holds the **font magnification** (1000 times the 'at size' of the font divided by its 'design size' (or just 1000 if there was no 'at' specification for the font)). Next comes the byte fnamlen, which is the number of characters in the **font name**, followed by the the **font name**, one ascii character per byte (right justified). Note that the font name includes a directory only if the font is not in the standard default library directory. From the definitions of the **parameters** of the PST command, note that the end of the **font definitions** is marked by a **font number** of -1 (which is *not* a legal font number). The four bytes following this phony **font number** constitute the **parameter** q, which is a **pointer** to the PST **command** (i.e., the beginning of the postamble). Next is a single byte **parameter** i (called the ID byte). Currently, the ID byte should always have a value of 1; it will be changed to 2 on the next incompatible release of .DVI format in 1990. Finally, there is some number (at least 4) of bytes whose value is 223 (base ten = '337 octal).

The idea of the q **pointer** at the end of the **postamble** is that a .DVI reading program can start at the end of the .DVI file, skipping backwards over the 223's, until it finds the ID byte. Then it can back up 4 bytes, read q, and then do a random seek to that byte number within the .DVI file. Now the **postamble** can be read from start to finish, while storing away the names and magnifications of all the fonts. Now the program can jump to the start of the .DVI file and read it sequentially. The reason for reading the **postamble** first is that to figure where the characters on a page go, the .DVI reading program must know the widths of the characters (see the VERTCHAR commands' description above). To find the widths, the .DVI reader must know the names of the fonts so it can get their widths from a TFM or VNT (or some other kind of font) file. But TEX can't put out all the font names until the end of the .DVI file because new fonts can appear anywhere in the TEX job. If **font definitions** were scattered throughout the .DVI file, then a spooler that read .DVI files would have to read all the pages of the .DVI file, even if the user only wanted the last page printed. The decision to put the **font definitions** in the **postamble** was based on these considerations, and the fact that just about any reasonable systems language allows random access. Unfortunately, standard PASCAL does not offer this feature. If it is absolutely necessary for a .DVI reading program to be written in standard PASCAL, then it either must make two passes over the .DVI file, or TEX must be doctored to output *two* files: the regular .DVI file, plus a PST file, which contains only the **postamble**. So far, there have been no reports of any installation of TEX that required this kind of kludge.

A few words on magnification: If you have a TEX document that does not mention any 'true' dimensions, then if you change just its `\magnify` statement, the .DVI file produced by TEX will change in just *one* place—the word in the postamble that records the requested magnification. The idea is that any spooler that reads the .DVI file will multiply *all* dimensions in the .DVI file by the magnification, thus the default magnification in the .DVI file may be easily overridden at spooling time. So, if the document specifies `\magnify{1200}`, a `\vskip 34cm` will be recorded in the .DVI file as $.34 \times 10^7$ rsu's of white space, but the spooler will multipy this by 1.2, making 40.8 centimeters of white space on output. If the user tells the spooler to use a magnification of 1000 rather than the 1200 in the .DVI file, then the output will have 34cm of white space. If a dimension in the document is specified as being 'true', then TEX divides the distance specified by the prevailing magnification, so that when a spooler looks at the .DVI file and multiplies by the magnification, it gets back the original distance. So, if we `\vskip 24truecm` while the magnification is 1200, TEX puts out .DVI commands that specifies 20 centimeters of white space. An output spooler that reads this .DVI file then puts $20 \times 1.2 = 24$cm of white space on its output. Of course, 'true' dimensions will come out 'false' if the spooler is told to override the magnification.

Font magnification goes one step further. Assume for a moment that the overall magnification is 1000. Now, if a TEX job specifies `\font A=CMR10 at 15pt`, say, that font's magnification is recorded as 1500 in its **font definition**. When a spooler reads this .DVI file, it will try to use the file CMR10.150VNT (or CMR10.150ANT, depending on the device), which is just like CMR10.100VNT, but the dimensions of all its characters were multiplied by 1.5 before they were digitized. An uppercase 'W' in CMR10 is 10pt wide, but CMR10 at 15pt has a 15pt wide 'W', so after VERTCHAR87 is seen, **horisontal coordinate** is increased by (15pt) $\times$ (254000rsu/72.27pt). Overall magnification is taken into account after all other calculations; for example, at magnification 1200 the font CMR10.120VNT would be used. Note that if the user had asked for `cmr10 at 15truept`, the factors would cancel out so that CMR10.150VNT would be the font chosen regardless of magnification. The magnification factor is given times 100 in the font file name so that roundoff error due to several multiplications will not affect the search for a font with characters of the right size. This convention about font file names is merely a suggestion, of course, it is not part of the .DVI format per se.

. . . . . .

Appendix: Comparison between version 0 and version 1.

Note that .DVI files have an ID byte at the end of the **postamble**, which tells what version they are. The changes since version 0 are:

DVI files now use the *upper* bits in a word on machines whose word size isn't evenly divisible by 8. The BOP command has *ten* \counter parameters. The size of rsu's has changed to be 10^{-7}meter. The postamble has changed to include overall magnification as well as a fraction that allows use of non-rsu dimensions. Font checksum and magnification are new, as is the convention about default directory name. Font descriptions in the postamble give the length of font names rather than delimiting them with a quoting character. The old zero ID byte is now a one.

Some ideas for version 2.

Although 1990 is still a ways off, we are currently expecting that version 2 of .DVI files will differ in the following ways:

The ID byte will be 2. The q bytes of the postamble will be preceded by '*s*[2]' where *s* is the maximum stack depth (excess of pushes over pops) needed to process this file.

* * * * * * * * * * *

SOME FEEDBACK FROM PTEX INSTALLATIONS

Ignacio Zabala

The Pascal version of TEX was designed and written with the intent to generate a transportable program. Nevertheless, given the characteristics of the TEX system, some special assumptions had to be made about the Pascal environment in which PTEX was to be installed. Essentially, the requirements are:

- The system should have enough addressable memory to store the large arrays employed by PTEX (about 128K words of 32 bits).
- The compiler should be able to really pack fields of a PACKED RECORD and overlap multiple variants of packed records. If this requisite is not satisfied, PTEX will require at least four times as much memory.
- The compiler should be able to handle large case statements (say over 64 actual cases in a range [–500..500]) and have a default case (this is non-standard in Pascal but available in most compilers).