

13. Device-independent file format. Before we get into the details of `DVItype`, we need to know exactly what DVI files are. The form of such files was designed by David R. Fuchs in 1979. Almost any reasonable typesetting device can be driven by a program that takes DVI files as input, and dozens of such DVI-to-whatever programs have been written. Thus, it is possible to print the output of document compilers like `TEX` on many different kinds of equipment.

A DVI file is a stream of 8-bit bytes, which may be regarded as a series of commands in a machine-like language. The first byte of each command is the operation code, and this code is followed by zero or more bytes that provide parameters to the command. The parameters themselves may consist of several consecutive bytes; for example, the `'set_rule'` command has two parameters, each of which is four bytes long. Parameters are usually regarded as nonnegative integers; but four-byte-long parameters, and shorter parameters that denote distances, can be either positive or negative. Such parameters are given in two's complement notation. For example, a two-byte-long distance parameter has a value between -2^{15} and $2^{15} - 1$.

A DVI file consists of a “preamble,” followed by a sequence of one or more “pages,” followed by a “postamble.” The preamble is simply a `pre` command, with its parameters that define the dimensions used in the file; this must come first. Each “page” consists of a `bop` command, followed by any number of other commands that tell where characters are to be placed on a physical page, followed by an `epo` command. The pages appear in the order that they were generated, not in any particular numerical order. If we ignore `nop` commands and `fnt_def` commands (which are allowed between any two commands in the file), each `epo` command is immediately followed by a `bop` command, or by a `post` command; in the latter case, there are no more pages in the file, and the remaining bytes form the postamble. Further details about the postamble will be explained later.

Some parameters in DVI commands are “pointers.” These are four-byte quantities that give the location number of some other byte in the file; the first byte is number 0, then comes number 1, and so on. For example, one of the parameters of a `bop` command points to the previous `bop`; this makes it feasible to read the pages in backwards order, in case the results are being directed to a device that stacks its output face up. Suppose the preamble of a DVI file occupies bytes 0 to 99. Now if the first page occupies bytes 100 to 999, say, and if the second page occupies bytes 1000 to 1999, then the `bop` that starts in byte 1000 points to 100 and the `bop` that starts in byte 2000 points to 1000. (The very first `bop`, i.e., the one that starts in byte 100, has a pointer of -1 .)

14. The DVI format is intended to be both compact and easily interpreted by a machine. Compactness is achieved by making most of the information implicit instead of explicit. When a DVI-reading program reads the commands for a page, it keeps track of several quantities: (a) The current font f is an integer; this value is changed only by `fnt` and `fnt_num` commands. (b) The current position on the page is given by two numbers called the horizontal and vertical coordinates, h and v . Both coordinates are zero at the upper left corner of the page; moving to the right corresponds to increasing the horizontal coordinate, and moving down corresponds to increasing the vertical coordinate. Thus, the coordinates are essentially Cartesian, except that vertical directions are flipped; the Cartesian version of (h, v) would be $(h, -v)$. (c) The current spacing amounts are given by four numbers $w, x, y,$ and z , where w and x are used for horizontal spacing and where y and z are used for vertical spacing. (d) There is a stack containing (h, v, w, x, y, z) values; the DVI commands `push` and `pop` are used to change the current level of operation. Note that the current font f is not pushed and popped; the stack contains only information about positioning.

The values of $h, v, w, x, y,$ and z are signed integers having up to 32 bits, including the sign. Since they represent physical distances, there is a small unit of measurement such that increasing h by 1 means moving a certain tiny distance to the right. The actual unit of measurement is variable, as explained below.

15. Here is a list of all the commands that may appear in a DVI file. Each command is specified by its symbolic name (e.g., *bop*), its opcode byte (e.g., 139), and its parameters (if any). The parameters are followed by a bracketed number telling how many bytes they occupy; for example, ‘*p*[4]’ means that parameter *p* is four bytes long.

- set_char_0* 0. Typeset character number 0 from font *f* such that the reference point of the character is at (*h*, *v*). Then increase *h* by the width of that character. Note that a character may have zero or negative width, so one cannot be sure that *h* will advance after this command; but *h* usually does increase.
- set_char_1* through *set_char_127* (opcodes 1 to 127). Do the operations of *set_char_0*; but use the character whose number matches the opcode, instead of character 0.
- set1* 128 *c*[1]. Same as *set_char_0*, except that character number *c* is typeset. T_EX82 uses this command for characters in the range $128 \leq c < 256$.
- set2* 129 *c*[2]. Same as *set1*, except that *c* is two bytes long, so it is in the range $0 \leq c < 65536$. T_EX82 never uses this command, which is intended for processors that deal with oriental languages; but DVItype will allow character codes greater than 255, assuming that they all have the same width as the character whose code is $c \bmod 256$.
- set3* 130 *c*[3]. Same as *set1*, except that *c* is three bytes long, so it can be as large as $2^{24} - 1$.
- set4* 131 *c*[4]. Same as *set1*, except that *c* is four bytes long, possibly even negative. Imagine that.
- set_rule* 132 *a*[4] *b*[4]. Typeset a solid black rectangle of height *a* and width *b*, with its bottom left corner at (*h*, *v*). Then set $h \leftarrow h + b$. If either $a \leq 0$ or $b \leq 0$, nothing should be typeset. Note that if $b < 0$, the value of *h* will decrease even though nothing else happens. Programs that typeset from DVI files should be careful to make the rules line up carefully with digitized characters, as explained in connection with the *rule_pixels* subroutine below.
- put1* 133 *c*[1]. Typeset character number *c* from font *f* such that the reference point of the character is at (*h*, *v*). (The ‘put’ commands are exactly like the ‘set’ commands, except that they simply put out a character or a rule without moving the reference point afterwards.)
- put2* 134 *c*[2]. Same as *set2*, except that *h* is not changed.
- put3* 135 *c*[3]. Same as *set3*, except that *h* is not changed.
- put4* 136 *c*[4]. Same as *set4*, except that *h* is not changed.
- put_rule* 137 *a*[4] *b*[4]. Same as *set_rule*, except that *h* is not changed.
- nop* 138. No operation, do nothing. Any number of *nop*’s may occur between DVI commands, but a *nop* cannot be inserted between a command and its parameters or between two parameters.
- bop* 139 *c*₀[4] *c*₁[4] ... *c*₉[4] *p*[4]. Beginning of a page: Set $(h, v, w, x, y, z) \leftarrow (0, 0, 0, 0, 0, 0)$ and set the stack empty. Set the current font *f* to an undefined value. The ten *c*_{*i*} parameters can be used to identify pages, if a user wants to print only part of a DVI file; T_EX82 gives them the values of $\backslash\text{count}0 \dots \backslash\text{count}9$ at the time $\backslash\text{shipout}$ was invoked for this page. The parameter *p* points to the previous *bop* command in the file, where the first *bop* has $p = -1$.
- eop* 140. End of page: Print what you have read since the previous *bop*. At this point the stack should be empty. (The DVI-reading programs that drive most output devices will have kept a buffer of the material that appears on the page that has just ended. This material is largely, but not entirely, in order by *v* coordinate and (for fixed *v*) by *h* coordinate; so it usually needs to be sorted into some order that is appropriate for the device in question. DVItype does not do such sorting.)
- push* 141. Push the current values of (h, v, w, x, y, z) onto the top of the stack; do not change any of these values. Note that *f* is not pushed.
- pop* 142. Pop the top six values off of the stack and assign them to (h, v, w, x, y, z) . The number of pops should never exceed the number of pushes, since it would be highly embarrassing if the stack were empty at the time of a *pop* command.
- right1* 143 *b*[1]. Set $h \leftarrow h + b$, i.e., move right *b* units. The parameter is a signed number in two’s complement notation, $-128 \leq b < 128$; if $b < 0$, the reference point actually moves left.

- right2* 144 *b*[2]. Same as *right1*, except that *b* is a two-byte quantity in the range $-32768 \leq b < 32768$.
- right3* 145 *b*[3]. Same as *right1*, except that *b* is a three-byte quantity in the range $-2^{23} \leq b < 2^{23}$.
- right4* 146 *b*[4]. Same as *right1*, except that *b* is a four-byte quantity in the range $-2^{31} \leq b < 2^{31}$.
- w0* 147. Set $h \leftarrow h + w$; i.e., move right *w* units. With luck, this parameterless command will usually suffice, because the same kind of motion will occur several times in succession; the following commands explain how *w* gets particular values.
- w1* 148 *b*[1]. Set $w \leftarrow b$ and $h \leftarrow h + b$. The value of *b* is a signed quantity in two's complement notation, $-128 \leq b < 128$. This command changes the current *w* spacing and moves right by *b*.
- w2* 149 *b*[2]. Same as *w1*, but *b* is a two-byte-long parameter, $-32768 \leq b < 32768$.
- w3* 150 *b*[3]. Same as *w1*, but *b* is a three-byte-long parameter, $-2^{23} \leq b < 2^{23}$.
- w4* 151 *b*[4]. Same as *w1*, but *b* is a four-byte-long parameter, $-2^{31} \leq b < 2^{31}$.
- x0* 152. Set $h \leftarrow h + x$; i.e., move right *x* units. The 'x' commands are like the 'w' commands except that they involve *x* instead of *w*.
- x1* 153 *b*[1]. Set $x \leftarrow b$ and $h \leftarrow h + b$. The value of *b* is a signed quantity in two's complement notation, $-128 \leq b < 128$. This command changes the current *x* spacing and moves right by *b*.
- x2* 154 *b*[2]. Same as *x1*, but *b* is a two-byte-long parameter, $-32768 \leq b < 32768$.
- x3* 155 *b*[3]. Same as *x1*, but *b* is a three-byte-long parameter, $-2^{23} \leq b < 2^{23}$.
- x4* 156 *b*[4]. Same as *x1*, but *b* is a four-byte-long parameter, $-2^{31} \leq b < 2^{31}$.
- down1* 157 *a*[1]. Set $v \leftarrow v + a$, i.e., move down *a* units. The parameter is a signed number in two's complement notation, $-128 \leq a < 128$; if *a* < 0, the reference point actually moves up.
- down2* 158 *a*[2]. Same as *down1*, except that *a* is a two-byte quantity in the range $-32768 \leq a < 32768$.
- down3* 159 *a*[3]. Same as *down1*, except that *a* is a three-byte quantity in the range $-2^{23} \leq a < 2^{23}$.
- down4* 160 *a*[4]. Same as *down1*, except that *a* is a four-byte quantity in the range $-2^{31} \leq a < 2^{31}$.
- y0* 161. Set $v \leftarrow v + y$; i.e., move down *y* units. With luck, this parameterless command will usually suffice, because the same kind of motion will occur several times in succession; the following commands explain how *y* gets particular values.
- y1* 162 *a*[1]. Set $y \leftarrow a$ and $v \leftarrow v + a$. The value of *a* is a signed quantity in two's complement notation, $-128 \leq a < 128$. This command changes the current *y* spacing and moves down by *a*.
- y2* 163 *a*[2]. Same as *y1*, but *a* is a two-byte-long parameter, $-32768 \leq a < 32768$.
- y3* 164 *a*[3]. Same as *y1*, but *a* is a three-byte-long parameter, $-2^{23} \leq a < 2^{23}$.
- y4* 165 *a*[4]. Same as *y1*, but *a* is a four-byte-long parameter, $-2^{31} \leq a < 2^{31}$.
- z0* 166. Set $v \leftarrow v + z$; i.e., move down *z* units. The 'z' commands are like the 'y' commands except that they involve *z* instead of *y*.
- z1* 167 *a*[1]. Set $z \leftarrow a$ and $v \leftarrow v + a$. The value of *a* is a signed quantity in two's complement notation, $-128 \leq a < 128$. This command changes the current *z* spacing and moves down by *a*.
- z2* 168 *a*[2]. Same as *z1*, but *a* is a two-byte-long parameter, $-32768 \leq a < 32768$.
- z3* 169 *a*[3]. Same as *z1*, but *a* is a three-byte-long parameter, $-2^{23} \leq a < 2^{23}$.
- z4* 170 *a*[4]. Same as *z1*, but *a* is a four-byte-long parameter, $-2^{31} \leq a < 2^{31}$.
- fnt_num_0* 171. Set $f \leftarrow 0$. Font 0 must previously have been defined by a *fnt_def* instruction, as explained below.
- fnt_num_1* through *fnt_num_63* (opcodes 172 to 234). Set $f \leftarrow 1, \dots, f \leftarrow 63$, respectively.
- fnt1* 235 *k*[1]. Set $f \leftarrow k$. T_EX82 uses this command for font numbers in the range $64 \leq k < 256$.
- fnt2* 236 *k*[2]. Same as *fnt1*, except that *k* is two bytes long, so it is in the range $0 \leq k < 65536$. T_EX82 never generates this command, but large font numbers may prove useful for specifications of color or texture, or they may be used for special fonts that have fixed numbers in some external coding scheme.

fnt3 237 $k[3]$. Same as *fnt1*, except that k is three bytes long, so it can be as large as $2^{24} - 1$.

fnt4 238 $k[4]$. Same as *fnt1*, except that k is four bytes long; this is for the really big font numbers (and for the negative ones).

xxx1 239 $k[1]$ $x[k]$. This command is undefined in general; it functions as a $(k+2)$ -byte *nop* unless special DVI-reading programs are being used. T_EX82 generates *xxx1* when a short enough `\special` appears, setting k to the number of bytes being sent. It is recommended that x be a string having the form of a keyword followed by possible parameters relevant to that keyword.

xxx2 240 $k[2]$ $x[k]$. Like *xxx1*, but $0 \leq k < 65536$.

xxx3 241 $k[3]$ $x[k]$. Like *xxx1*, but $0 \leq k < 2^{24}$.

xxx4 242 $k[4]$ $x[k]$. Like *xxx1*, but k can be ridiculously large. T_EX82 uses *xxx4* when *xxx1* would be incorrect.

fnt_def1 243 $k[1]$ $c[4]$ $s[4]$ $d[4]$ $a[1]$ $l[1]$ $n[a+l]$. Define font k , where $0 \leq k < 256$; font definitions will be explained shortly.

fnt_def2 244 $k[2]$ $c[4]$ $s[4]$ $d[4]$ $a[1]$ $l[1]$ $n[a+l]$. Define font k , where $0 \leq k < 65536$.

fnt_def3 245 $k[3]$ $c[4]$ $s[4]$ $d[4]$ $a[1]$ $l[1]$ $n[a+l]$. Define font k , where $0 \leq k < 2^{24}$.

fnt_def4 246 $k[4]$ $c[4]$ $s[4]$ $d[4]$ $a[1]$ $l[1]$ $n[a+l]$. Define font k , where $-2^{31} \leq k < 2^{31}$.

pre 247 $i[1]$ $num[4]$ $den[4]$ $mag[4]$ $k[1]$ $x[k]$. Beginning of the preamble; this must come at the very beginning of the file. Parameters i , num , den , mag , k , and x are explained below.

post 248. Beginning of the postamble, see below.

post_post 249. Ending of the postamble, see below.

Commands 250–255 are undefined at the present time.

```

16. define set_char_0 = 0 { typeset character 0 and move right }
define set1 = 128 { typeset a character and move right }
define set_rule = 132 { typeset a rule and move right }
define put1 = 133 { typeset a character }
define put_rule = 137 { typeset a rule }
define nop = 138 { no operation }
define bop = 139 { beginning of page }
define eop = 140 { ending of page }
define push = 141 { save the current positions }
define pop = 142 { restore previous positions }
define right1 = 143 { move right }
define w0 = 147 { move right by w }
define w1 = 148 { move right and set w }
define x0 = 152 { move right by x }
define x1 = 153 { move right and set x }
define down1 = 157 { move down }
define y0 = 161 { move down by y }
define y1 = 162 { move down and set y }
define z0 = 166 { move down by z }
define z1 = 167 { move down and set z }
define fnt_num_0 = 171 { set current font to 0 }
define fnt1 = 235 { set current font }
define xxx1 = 239 { extension to DVI primitives }
define xxx4 = 242 { potentially long extension to DVI primitives }
define fnt_def1 = 243 { define the meaning of a font number }
define pre = 247 { preamble }
define post = 248 { postamble beginning }
define post_post = 249 { postamble ending }
define undefined_commands  $\equiv$  250, 251, 252, 253, 254, 255

```

17. The preamble contains basic information about the file as a whole. As stated above, there are six parameters:

$$i[1] \text{ num}[4] \text{ den}[4] \text{ mag}[4] \text{ k}[1] \text{ x}[k].$$

The *i* byte identifies DVI format; currently this byte is always set to 2. (The value *i* = 3 is currently used for an extended format that allows a mixture of right-to-left and left-to-right typesetting. Some day we will set *i* = 4, when DVI format makes another incompatible change—perhaps in the year 2048.)

The next two parameters, *num* and *den*, are positive integers that define the units of measurement; they are the numerator and denominator of a fraction by which all dimensions in the DVI file could be multiplied in order to get lengths in units of 10^{-7} meters. (For example, there are exactly 7227 T_EX points in 254 centimeters, and T_EX82 works with scaled points where there are 2^{16} sp in a point, so T_EX82 sets *num* = 25400000 and *den* = $7227 \cdot 2^{16} = 473628672$.)

The *mag* parameter is what T_EX82 calls `\mag`, i.e., 1000 times the desired magnification. The actual fraction by which dimensions are multiplied is therefore $mn/1000d$. Note that if a T_EX source document does not call for any ‘true’ dimensions, and if you change it only by specifying a different `\mag` setting, the DVI file that T_EX creates will be completely unchanged except for the value of *mag* in the preamble and postamble. (Fancy DVI-reading programs allow users to override the *mag* setting when a DVI file is being printed.)

Finally, *k* and *x* allow the DVI writer to include a comment, which is not interpreted further. The length of comment *x* is *k*, where $0 \leq k < 256$.

```

define id_byte = 2 { identifies the kind of DVI files described here }

```

18. Font definitions for a given font number k contain further parameters

$$c[4] s[4] d[4] a[1] l[1] n[a + l].$$

The four-byte value c is the check sum that $\text{T}_{\text{E}}\text{X}$ (or whatever program generated the DVI file) found in the TFM file for this font; c should match the check sum of the font found by programs that read this DVI file.

Parameter s contains a fixed-point scale factor that is applied to the character widths in font k ; font dimensions in TFM files and other font files are relative to this quantity, which is always positive and less than 2^{27} . It is given in the same units as the other dimensions of the DVI file. Parameter d is similar to s ; it is the “design size,” and (like s) it is given in DVI units. Thus, font k is to be used at $\text{mag} \cdot s/1000d$ times its normal size.

The remaining part of a font definition gives the external name of the font, which is an ASCII string of length $a + l$. The number a is the length of the “area” or directory, and l is the length of the font name itself; the standard local system font area is supposed to be used when $a = 0$. The n field contains the area in its first a bytes.

Font definitions must appear before the first use of a particular font number. Once font k is defined, it must not be defined again; however, we shall see below that font definitions appear in the postamble as well as in the pages, so in this sense each font number is defined exactly twice, if at all. Like *nop* commands, font definitions can appear before the first *bop*, or between an *ep* and a *bop*.

19. The last page in a DVI file is followed by ‘*post*’; this command introduces the postamble, which summarizes important facts that $\text{T}_{\text{E}}\text{X}$ has accumulated about the file, making it possible to print subsets of the data with reasonable efficiency. The postamble has the form

$$\begin{aligned} & \textit{post} p[4] \textit{num}[4] \textit{den}[4] \textit{mag}[4] l[4] u[4] s[2] t[2] \\ & \langle \textit{font definitions} \rangle \\ & \textit{post_post} q[4] i[1] 223's[\geq 4] \end{aligned}$$

Here p is a pointer to the final *bop* in the file. The next three parameters, \textit{num} , \textit{den} , and \textit{mag} , are duplicates of the quantities that appeared in the preamble.

Parameters l and u give respectively the height-plus-depth of the tallest page and the width of the widest page, in the same units as other dimensions of the file. These numbers might be used by a DVI-reading program to position individual “pages” on large sheets of film or paper; however, the standard convention for output on normal size paper is to position each page so that the upper left-hand corner is exactly one inch from the left and the top. Experience has shown that it is unwise to design DVI-to-printer software that attempts cleverly to center the output; a fixed position of the upper left corner is easiest for users to understand and to work with. Therefore l and u are often ignored.

Parameter s is the maximum stack depth (i.e., the largest excess of *push* commands over *pop* commands) needed to process this file. Then comes t , the total number of pages (*bop* commands) present.

The postamble continues with font definitions, which are any number of *fnt_def* commands as described above, possibly interspersed with *nop* commands. Each font number that is used in the DVI file must be defined exactly twice: Once before it is first selected by a *fnt* command, and once in the postamble.

20. The last part of the postamble, following the *post_post* byte that signifies the end of the font definitions, contains *q*, a pointer to the *post* command that started the postamble. An identification byte, *i*, comes next; this currently equals 2, as in the preamble.

The *i* byte is followed by four or more bytes that are all equal to the decimal number 223 (i.e., *'337* in octal). \TeX puts out four to seven of these trailing bytes, until the total length of the file is a multiple of four bytes, since this works out best on machines that pack four bytes per word; but any number of 223's is allowed, as long as there are at least four of them. In effect, 223 is a sort of signature that is added at the very end.

This curious way to finish off a DVI file makes it feasible for DVI-reading programs to find the postamble first, on most computers, even though \TeX wants to write the postamble last. Most operating systems permit random access to individual words or bytes of a file, so the DVI reader can start at the end and skip backwards over the 223's until finding the identification byte. Then it can back up four bytes, read *q*, and move to byte *q* of the file. This byte should, of course, contain the value 248 (*post*); now the postamble can be read, so the DVI reader discovers all the information needed for typesetting the pages. Note that it is also possible to skip through the DVI file at reasonably high speed to locate a particular page, if that proves desirable. This saves a lot of time, since DVI files used in production jobs tend to be large.

Unfortunately, however, standard Pascal does not include the ability to access a random position in a file, or even to determine the length of a file. Almost all systems nowadays provide the necessary capabilities, so DVI format has been designed to work most efficiently with modern operating systems. As noted above, \DVItype will limit itself to the restrictions of standard Pascal if *random_reading* is defined to be *false*.