The DVItype processor

(Version 3.6, December 1995)

The preparation of this report was supported in part by the National Science Foundation under grants IST-8201926 and MCS-8300984, and by the System Development Foundation. ‘\TeX’ is a trademark of the American Mathematical Society.
1* Introduction. The DViType utility program reads binary device-independent ("DVI") files that are produced by document compilers such as TeX, and converts them into symbolic form. This program has two chief purposes: (1) It can be used to determine whether a DVI file is valid or invalid, when diagnosing compiler errors; and (2) it serves as an example of a program that reads DVI files correctly, for system programmers who are developing DVI-related software.

Goal number (2) needs perhaps a bit more explanation. Programs for typesetting need to be especially careful about how they do arithmetic; if rounding errors accumulate, margins won’t be straight, vertical rules won’t line up, and so on. But if rounding is done everywhere, even in the midst of words, there will be uneven spacing between the letters, and that looks bad. Human eyes notice differences of a thousandth of an inch in the positioning of lines that are close together; on low resolution devices, where rounding produces effects four times as great as this, the problem is especially critical. Experience has shown that unusual care is needed even on high-resolution equipment; for example, a mistake in the sixth significant hexadecimal place of a constant once led to a difficult-to-find bug in some software for the Alphatype CRS, which has a resolution of 5333 pixels per inch (make that 5333.33333333 pixels per inch). The document compilers that generate DVI files make certain assumptions about the arithmetic that will be used by DVI-reading software, and if these assumptions are violated the results will be of inferior quality. Therefore the present program is intended as a guide to proper procedure in the critical places where a bit of subtlety is involved.

The first DViType program was designed by David Fuchs in 1979, and it went through several versions on different computers as the format of DVI files was evolving to its present form. Peter Breitenlohner helped with the latest revisions.

The banner string defined here should be changed whenever DViType gets modified.

```
define my_name ≡ 'dvitype'
define banner ≡ 'This is DViType, Version 3.6'  { printed when the program starts }
```

3* The binary input comes from dvi_file, and the symbolic output is written on Pascal’s standard output file. The term print is used instead of write when this program writes on output, so that all such output could easily be redirected if desired.

```
define print(#) ≡ write(stdout, #)
define print_ln(#) ≡ write_ln(stdout, #)
```

```
program DVI_type(dvi_file, output);
  label { Labels in the outer block 4* }
  const { Constants in the outer block 5* }
  type { Types in the outer block 8* }
  var { Globals in the outer block 10 }
    { Define parse_arguments 112* }
  procedure initialize;  { this procedure gets things started properly }
    var i: integer;  { loop index for initializations }
    begin kpse_set_program_name(argv[0], my_name); parse_arguments; print(banner);
      print_ln(version_string);  { Set initial values 11 }
    end;

4* Label done is used when stopping normally.
```

```
define done = 30  { go here when finished with a subtask }
```

This code is used in section 3*.
The following parameters can be changed at compile time to extend or reduce DVItype’s capacity.

\[
\begin{align*}
\text{max\_fonts} & = 500; & \{\text{maximum number of distinct fonts per DVI file}\} \\
\text{max\_widths} & = 25000; & \{\text{maximum number of different characters among all fonts}\} \\
\text{line\_length} & = 79; & \{\text{bracketed lines of output will be at most this long}\} \\
\text{stack\_size} & = 100; & \{\text{DVI files shouldn’t push beyond this depth}\} \\
\text{name\_size} & = 10000; & \{\text{total length of all font file names}\}
\end{align*}
\]

This code is used in section 3*.

If the DVI file is badly malformed, the whole process must be aborted; DVItype will give up, after issuing an error message about the symptoms that were noticed.

Such errors might be discovered inside of subroutines inside of subroutines, so a procedure called \texttt{jump\_out} has been introduced.

\begin{verbatim}
define \texttt{jump\_out} \equiv \texttt{uexit}(1) 
define \texttt{abort}(\#) \equiv 
    \begin{verbatim}
    begin \texttt{write\_ln}(\texttt{stderr},\#); \texttt{jump\_out}; 
    end
\end{verbatim} 
define \texttt{bad\_dvi}(\#) \equiv \texttt{abort}(\texttt{´Bad\_DVI\_file:´},\#,\texttt{´!´})
\end{verbatim}
§ 8* The character set. Like all programs written with the WEB system, DVItype can be used with any character set. But it uses ASCII code internally, because the programming for portable input-output is easier when a fixed internal code is used, and because DVI files use ASCII code for file names and certain other strings.

The next few sections of DVItype have therefore been copied from the analogous ones in the WEB system routines. They have been considerably simplified, since DVItype need not deal with the controversial ASCII codes less than ‘40 or greater than ‘176. If such codes appear in the DVI file, they will be printed as question marks.

\[
\langle \text{Types in the outer block 8*} \rangle \equiv
\]
\[
\text{ASCII code} = 0 \ldots 255; \quad \{ \text{a subrange of the integers} \}
\]

See also sections 9* and 21.

This code is used in section 3*.

9* The original Pascal compiler was designed in the late 60s, when six-bit character sets were common, so it did not make provision for lower case letters. Nowadays, of course, we need to deal with both upper and lower case alphabets in a convenient way, especially in a program like DVItype. So we shall assume that the Pascal system being used for DVItype has a character set containing at least the standard visible characters of ASCII code ("!" through "~").

Some Pascal compilers use the original name char for the data type associated with the characters in text files, while other Pascals consider char to be a 64-element subrange of a larger data type that has some other name. In order to accommodate this difference, we shall use the name text_char to stand for the data type of the characters in the output file. We shall also assume that text_char consists of the elements chr(first_text_char) through chr(last_text_char), inclusive. The following definitions should be adjusted if necessary.

\[
\text{define text_char} \equiv \text{ASCII code} \quad \{ \text{the data type of characters in text files} \}
\]
\[
\text{define first_text_char} = 0 \quad \{ \text{ordinal number of the smallest element of text_char} \}
\]
\[
\text{define last_text_char} = 255 \quad \{ \text{ordinal number of the largest element of text_char} \}
\]

\[
\langle \text{Types in the outer block 8*} \rangle +\equiv
\]
\[
\text{text_file} = \text{packed file of text_char};
\]
To prepare these files for input, we *reset* them. An extension of Pascal is needed in the case of `tfm_file`, since we want to associate it with external files whose names are specified dynamically (i.e., not known at compile time). The following code assumes that ‘`reset(f, s)`’ does this, when `f` is a file variable and `s` is a string variable that specifies the file name. If `eof(f)` is true immediately after `reset(f, s)` has acted, we assume that no file named `s` is accessible.

```pascal
procedure open_dvi_file; { prepares to read packed bytes in dvi_file }
  begin resetbin(dvi_file, dvi_name); cur_loc ← 0;
  end;

procedure open_tfm_file; { prepares to read packed bytes in tfm_file }
  var full_name: ↑char;
  begin full_name ← kpse_find_tfm(cur_name);
    if full_name then
      begin tfm_file ← fopen(full_name, FOPEN_RBIN_MODE);
      end
    else begin tfm_file ← nil;
      end;
  end;
```

If you looked carefully at the preceding code, you probably asked, “What are `cur_loc` and `cur_name`?” Good question. They’re global variables: `cur_loc` is the number of the byte about to be read next from `dvi_file`, and `cur_name` is a string variable that will be set to the current font metric file name before `open_tfm_file` is called.

```pascal
〈Globals in the outer block 10〉 +≡
cur_loc: integer; { where we are about to look, in dvi_file }
cur_name: ↑char; { external name }
```

Finally we come to the routines that are used only if `random_reading` is true. The driver program below needs two such routines: `dvi_length` should compute the total number of bytes in `dvi_file`, possibly also causing `eof(dvi_file)` to be true; and `move_to_byte(n)` should position `dvi_file` so that the next `get_byte` will read byte `n`, starting with `n = 0` for the first byte in the file.

Such routines are, of course, highly system dependent. They are implemented here in terms of two assumed system routines called `set_pos` and `cur_pos`. The call `set_pos(f, n)` moves to item `n` in file `f`, unless `n` is negative or larger than the total number of items in `f`; in the latter case, `set_pos(f, n)` moves to the end of file `f`. The call `cur_pos(f)` gives the total number of items in `f`, if `eof(f)` is true; we use `cur_pos` only in such a situation.

```pascal
function dvi_length: integer;
  begin xfseek(dvi_file, 0, 2, dvi_name); cur_loc ← xftell(dvi_file, dvi_name); dvi_length ← cur_loc;
  end;

procedure move_to_byte(n : integer);
  begin xfseek(dvi_file, n, 0, dvi_name); cur_loc ← n;
  end;
```
The starting page specification is recorded in two global arrays called `start_count` and `start_there`. For example, ‘1.*.−5’ is represented by `start_there[0] = true, start_count[0] = 1, start_there[1] = false, start_there[2] = true, start_count[2] = −5`. We also set `start_vals = 2`, to indicate that count 2 was the last one mentioned. The other values of `start_count` and `start_there` are not important, in this example.

\[
\text{\texttt{(Globals in the outer block 10)}} \equiv \\
\text{\texttt{start_count: array [0 .. 9] of integer; \{ count values to select starting page \}}} \\
\text{\texttt{start_there: array [0 .. 9] of boolean; \{ is the start_count value relevant? \}}} \\
\text{\texttt{start_vals: 0 .. 9; \{ the last count considered significant \}}} \\
\text{\texttt{count: array [0 .. 9] of integer; \{ the count values on the current page \}}}
\]

Initializations are done sooner now.

No dialog.

During the dialog, `DVItype` will treat the first blank space in a line as the end of that line. Therefore `input_in` makes sure that there is always at least one blank space in `buffer`.

No dialog.

No dialog.

No dialog (50).

No dialog (51).

No dialog (52).

No dialog (53).

No dialog (54).

No dialog (55).
After the dialog is over, we print the options so that the user can see what DVI type thought was specified.

```plaintext
⟨Print all the selected options 56∗⟩≡
print_ln(˝Options selected:´);
for k ← 0 to start_vals do
  begin if start_there[k] then print(start_count[k] : 1)
  else print(˝*˝);
  if k < start_vals then print(˝.")
  else print_ln(˝*˝);
end;
printLn(˝Maximum number of pages =´, max_pages : 1);
pḍ(˝Output level =´, out_mode : 1);
case out_mode of
  errors_only: printLn(˝(showing bops, fonts, and error messages only)´);
  terse: printLn(˝(terse)´);
  mnemonics_only: printLn(˝(mnemonics)´);
  verbose: printLn(˝(verbose)´);
  the_works: if random_reading then printLn(˝(the works)´)
      else begin out_mode ← verbose; printLn(˝(the works: same as level 3 in this DVI)´);
      end;
end;
pḍ(˝Resolution =´); print_real(resolution, 12, 8); printLn(˝pixels per inch´);
if new_mag > 0 then
  begin print(˝New magnification factor =´); print_real(new_mag/1000.0, 8, 3); printLn(˝) end
```

This code is used in section 107∗.
The following subroutine does the necessary things when a \texttt{fnt\_def} command is being processed.

**procedure** \texttt{define\_font}(\texttt{e : integer}) \{ \texttt{e} is an external font number \}

\begin{verbatim}
var \texttt{f : 0 .. max\_fonts; p : integer; \{ length of the area/directory spec \}}
  \texttt{n : integer; \{ length of the font name proper \}}
  \texttt{c, q, d, m : integer; \{ check sum, scaled size, design size, magnification \}}
  \texttt{r : 0 .. name\_size; \{ current filename length \}}
  \texttt{j, k : 0 .. name\_size; \{ indices into names \}}
  \texttt{mismatch : boolean; \{ do names disagree? \}}

begin \texttt{if n = max\_fonts then}
  \texttt{abort('DVItype\_capacity\_exceeded\(max\_fonts=\', max\_fonts : 1, \'}');}
  \texttt{font\_num[nf] ← e; f ← 0;}
  \texttt{while font\_num[f] \neq e do incr(f);}
  \texttt{(Read the font parameters into position for font \texttt{nf}, and print the font name \texttt{61});}
  \texttt{if \((out\_mode = the\_works) \land in\_postamble) \lor ((out\_mode < the\_works) \land \lnot in\_postamble) \texttt{then}}
    \texttt{begin \texttt{if f < nf then print\_ln('---this\_font\_was\_already\_defined!');}}
    \texttt{end \texttt{else begin if f = nf then print\_ln('---this\_font\_wasn't\_loaded\_before!');}}
    \texttt{end;}
  \texttt{if f = nf \texttt{then} \{ Load the new font, unless there are problems \texttt{62}\};}
  \texttt{else \{ Check that the current font definition matches the old one \texttt{60}; \}}
  \texttt{end;}
\end{verbatim}

\texttt{62* \{ Load the new font, unless there are problems \texttt{62}\} \equiv}

\begin{verbatim}
\texttt{begin \{ Move font name into the \texttt{cur\_name} string \texttt{66}\};}
  \texttt{open\_tfm;}
  \texttt{if \texttt{eof(tfm\_file)} \texttt{then} \texttt{print('---not\_loaded,\_TFM\_file\_can't\_be\_opened!')}}
  \texttt{else begin \{ \texttt{q} \leq 0 \lor (q \geq 1000000000) \texttt{then print('---not\_loaded,\_bad\_scale\(\', q : 1, \'}') \}}
    \texttt{else if \{ \texttt{d} \leq 0 \lor \texttt{d} \geq 1000000000 \texttt{then print('---not\_loaded,\_bad\_design\_size\(\', d : 1, \'}') \}}
    \texttt{else if in\_TFM(q) \texttt{then} \{ Finish loading the new font info \texttt{63}; \}}
    \texttt{end;}
  \texttt{if out\_mode = errors\_only \texttt{then} print\_ln('\_\_\_');}
  \texttt{if tfm\_file \texttt{then xfclose(tfm\_file, cur\_name); \{ should be the kpse\_find\_tfm result \}}}
  \texttt{free(cur\_name); \{ We \texttt{xmalloc'd} this before we got called. \}}
\end{verbatim}

This code is used in section \texttt{59*}.

\texttt{64* If p = 0, i.e., if no font directory has been specified, DVItype is supposed to use the default font directory, which is a system-dependent place where the standard fonts are kept. The string variable default\_directory contains the name of this area.}

Under Unix, users have a path searched for fonts, there’s no single default directory.

\texttt{65* (No initialization needs to be done. Keep this module to preserve numbering.)}
The string *cur_name* is supposed to be set to the external name of the TFM file for the current font. We do not impose a maximum limit here. It’s too bad there is a limit on the total length of all filenames, but it doesn’t seem worth reprogramming all that.

\[
\text{define } \text{name}_\text{start} \equiv \text{font}_\text{name}[\text{nf}] \\
\text{define } \text{name}_\text{end} \equiv \text{font}_\text{name}[\text{nf} + 1]
\]

(Move font name into the *cur_name* string \(66^\ast\)) \(\equiv\)

\[
\begin{align*}
\text{for } k \leftarrow \text{name}_\text{start} \text{ to } \text{name}_\text{end} \text{ do} \\
&\quad \text{begin } \text{cur}_\text{name}[k - \text{name}_\text{start}] \leftarrow \text{xchr}[\text{names}[k]]; \\
&\quad \text{end}; \\
&\quad \text{cur}_\text{name}[r] \leftarrow 0; \quad \text{Append null byte for C.}
\end{align*}
\]

This code is used in section \(62^\ast\).
Before we get into the details of *do_page*, it is convenient to consider a simpler routine that computes the first parameter of each opcode.

```c
#define four_cases(#) ≡ #, # + 1, # + 2, # + 3
#define eight_cases(#) ≡ four_cases(#), four_cases(# + 4)
#define sixteen_cases(#) ≡ eight_cases(#), eight_cases(# + 8)
#define thirty_two_cases(#) ≡ sixteen_cases(#), sixteen_cases(# + 16)
#define sixty_four_cases(#) ≡ thirty_two_cases(#), thirty_two_cases(# + 32)

function first_par(o : eight_bits): integer;
  begin case o of
    sixty_four_cases(set_char_0), sixty_four_cases(set_char_0 + 64): first_par ← o - set_char_0;
    set1, put1, fnt1, xxx1, fnt_def1: first_par ← get_byte;
    set1 + 1, put1 + 1, fnt1 + 1, xxx1 + 1, fnt_def1 + 1: first_par ← get_two_bytes;
    set1 + 2, put1 + 2, fnt1 + 2, xxx1 + 2, fnt_def1 + 2: first_par ← get_three_bytes;
    right1, w1, x1, down1, y1, z1: first_par ← signed_byte;
    right1 + 1, w1 + 1, x1 + 1, down1 + 1, y1 + 1, z1 + 1: first_par ← signed_pair;
    right1 + 2, w1 + 2, x1 + 2, down1 + 2, y1 + 2, z1 + 2: first_par ← signed_trio;
    set1 + 3, set_rule, put1 + 3, put_rule, right1 + 3, w1 + 3, x1 + 3, down1 + 3, y1 + 3, z1 + 3, fnt1 + 3,
      xxx1 + 3, fnt_def1 + 3: first_par ← signed_quad;
    nop, bop, cop, push, pop, pre, post, post_post, undefined_commands: first_par ← 0;
    w0: first_par ← w;
    x0: first_par ← x;
    y0: first_par ← y;
    z0: first_par ← z;
    sixty_four_cases(fnt_num_0): first_par ← o - fnt_num_0;
    othercases abort(‘internal_error’);
  endcases;
end;
```
Commands are broken down into “major” and “minor” categories: A major command is always shown in full, while a minor one is put into the buffer in abbreviated form. Minor commands, which account for the bulk of most DVI files, involve horizontal spacing and the typesetting of characters in a line; these are shown in full only if \texttt{out\_mode} $\geq$ \texttt{verbose}.

\begin{verbatim}
80* define show(#) ≡
    begin flush_text; showing ← true; print(a : 1, ¦:Una, #);
    if show_opcodes $\land$ (o $\geq$ 128) then print(¦Una, o : 1, ¦{})
end

define major(#) ≡
    if out_mode > errors_only then show(#)

define minor(#) ≡
    if out_mode > terse then
        begin showing ← true; print(a : 1, ¦:Una, #);
        if show_opcodes $\land$ (o $\geq$ 128) then print(¦Una, o : 1, ¦{})
    end

define error(#) ≡
    if ¬showing then show(#)
    else print(¦Una, #)
\end{verbatim}

(Translate the next command in the DVI file; \texttt{goto} 9999 with \texttt{do\_page} = \texttt{true} if it was \texttt{eop}; \texttt{goto} 9998 if premature termination is needed \texttt{80*})

\begin{verbatim}
80* begin a ← cur_loc; showing ← false; o ← get_byte; p ← first_par(o);
if eof(dvi_file) then bad_dvi(“the file ended prematurely”);
(Start translation of command \texttt{o} and \texttt{goto} the appropriate label to finish the job \texttt{81});

fin\_set: (Finish a command that either sets or puts a character, then \texttt{goto} \texttt{move\_right} or \texttt{done} \texttt{89});

fin\_rule: (Finish a command that either sets or puts a rule, then \texttt{goto} \texttt{move\_right} or \texttt{done} \texttt{90});

move\_right: (Finish a command that sets \texttt{h ← h + q}, then \texttt{goto} \texttt{done} \texttt{91});

show\_state: (Show the values of \texttt{ss}, \texttt{h}, \texttt{v}, \texttt{w}, \texttt{x}, \texttt{y}, \texttt{z}, \texttt{hh}, and \texttt{vv}; then \texttt{goto} \texttt{done} \texttt{93});

80* done: if showing then print_line(¦Una);
\end{verbatim}

This code is used in section \texttt{79}.
99* The scan_bop procedure reads DVI commands following the preamble or following eop, until finding either bop or the postamble.

⟨ Declare the procedure called scan_bop 99* ⟩ ≡

procedure scan_bop:
  var k: 0 . . 255; { command code }
  begin repeat if eof(dvi_file) then bad_dvi(´the_file Ended prematurely´);
    k ← get_byte;
    if (k ≥ fnt_def1) ∧ (k < fnt_def1 + 4) then
      begin define_font(first_par(k));
        if out_mode ≠ errors_only then print_ln(´\'\');
        k ← nop;
      end;
    until k ≠ nop;
  if k = post then in_postamble ← true
  else begin if k ≠ bop then bad_dvi(´byte\', cur_loc − 1 : 1, ´is not bop´);
    new_backpointer ← cur_loc − 1; incr(page_count);
    for k ← 0 to 9 do count[k] ← signed_quad;
    if signed_quad ≠ old_backpointer then
      begin print_ln(´backpointer\', cur_loc − 4 : 1, ´should be\', old_backpointer : 1, ´!´);
        old_backpointer ← new_backpointer;
      end;
  end;
This code is used in section 95.
107* The main program. Now we are ready to put it all together. This is where DVItype starts, and where it ends.

begin initialize; { get all variables initialized }
⟨ Print all the selected options 56* );
⟨ Process the preamble 109 );
if out_mode = the_works then { random_reading = true }
begin ⟨ Find the postamble, working back from the end 100 );
in_postamble ← true; read_postamble; in_postamble ← false;
⟨ Count the pages and move to the starting page 102 );
end;
skip_pages(false);
if ¬in_postamble then ⟨ Translate up to max_pages pages 111 );
if out_mode < the_works then
begin if ¬in_postamble then skip_pages(true);
if signed_quad ≠ old_backpointer then
print(ln(`backpointer_in_byte`, cur_loc ← 4 : 1 , `should_be`, old_backpointer : 1 , `!` ));
read_postamble;
end;
end.

110* The conversion factor conv is figured as follows: There are exactly \( n/d \) decimicrons per DVI unit, and 254000 decimicrons per inch, and resolution pixels per inch. Then we have to adjust this by the stated amount of magnification.

⟨ Compute the conversion factors 110* ) ≡
numerator ← signed_quad; denominator ← signed_quad;
if numerator ≤ 0 then bad_dvi(`numerator_is`, numerator : 1 );
if denominator ≤ 0 then bad_dvi(`denominator_is`, denominator : 1 );
print(ln(`numerator/denominator=`, numerator : 1 , `/`, denominator : 1 );
tfm_conv ← (25400000.0/numerator) * (denominator/473628672)/16.0;
conv ← (numerator/254000.0) * (resolution/denominator); mag ← signed_quad;
if new_mag > 0 then mag ← new_mag
else if mag ≤ 0 then bad_dvi(`magnification_is`, mag : 1 );
true_conv ← conv; conv ← true_conv * (mag/1000.0); print(`magnification=`, mag : 1 , `; `);
print_real(conv, 16, 8); print_ln(`pixels_per_DVI_unit` )

This code is used in section 109.
System-dependent changes. Parse a Unix-style command line.

Define \( \text{argument\_is}(#) \equiv (\text{strcmp}(\text{long\_options}[\text{option\_index}].\text{name}, #) = 0) \)

(Define \( \text{parse\_arguments} \))

\[
\text{procedure parse\_arguments;}
\]
\[
\text{const } n\_\text{options} = 8; \{ \text{Pascal won’t count array lengths for us. } \}
\]
\[
\text{var long\_options: array[0..n\_options] of } \text{getopt\_struct};
\]
\[
\text{getopt\_return\_val: integer; option\_index: c\_int\_type; current\_option: 0..n\_options; end\_num: } \text{\textasciitildechar;}
\]
\[
\{ \text{for page\_start} \}
\]
\[
\text{begin} \{ \text{Define the option table} \}
\]
\[
\text{repeat } \text{getopt\_return\_val} \leftarrow \text{getopt\_long\_only}(\text{argc, argv, ‘’, long\_options, address\_of\,(option\_index)});
\]
\[
\text{if } \text{getopt\_return\_val} = -1 \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{do\_nothing;} \{ \text{End of arguments; we exit the loop below. } \} \\
\text{end}
\end{align*}
\]
\[
\text{else if } \text{getopt\_return\_val} = "?" \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{usage(my\_name);} \\
\text{end}
\end{align*}
\]
\[
\text{else if } \text{argument\_is(‘help’)} \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{usage\_help(DVITYPE\_HELP, nil);} \\
\text{end}
\end{align*}
\]
\[
\text{else if } \text{argument\_is(‘version’)} \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{print\_version\_and\_exit(banner, nil, ‘D.E.\_Knuth’, nil);} \\
\text{end}
\end{align*}
\]
\[
\text{else if } \text{argument\_is(‘output\_level’)} \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{if } (\text{optarg}[0] < ‘0’ ) \lor (\text{optarg}[0] > ‘4’ ) \lor (\text{optarg}[1] \neq 0) \text{ then }
\end{align*}
\]
\[
\begin{align*}
\text{begin } & \text{write\_ln(stderr, ‘Value for --output\_level must be >=0 and <=4.’); } \\
\text{uexit}(1); \\
\text{end; }
\end{align*}
\]
\[
\begin{align*}
\text{out\_mode} & \leftarrow \text{optarg}[0] - ‘0’ ;
\end{align*}
\]
\[
\text{end}
\]
\[
\text{else if } \text{argument\_is(‘page\_start’)} \text{ then }
\]
\[
\begin{align*}
\text{begin} \{ \text{Determine the desired start\_count values from optarg} \}
\end{align*}
\]
\[
\text{else if } \text{argument\_is(‘max\_pages’)} \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{max\_pages} \leftarrow \text{atou(optarg);} \\
\text{end}
\end{align*}
\]
\[
\text{else if } \text{argument\_is(‘dpi’)} \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{resolution} \leftarrow \text{atof(optarg);} \\
\text{end}
\end{align*}
\]
\[
\text{else if } \text{argument\_is(‘magnification’)} \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{new\_mag} \leftarrow \text{atou(optarg);} \\
\text{end; } \{ \text{Else it was a flag; getopt has already done the assignment. } \}
\end{align*}
\]
\[
\text{until } \text{getopt\_return\_val} = -1; \{ \text{Now optind is the index of first non-option on the command line. } \}
\]
\[
\text{if } (\text{optind} + 1 \neq \text{argc}) \text{ then }
\]
\[
\begin{align*}
\text{begin } & \text{write\_ln(stderr, my\_name, ‘: ‘Need exactly one file argument.’); } \text{usage(my\_name);} \\
\text{end; }
\end{align*}
\]
\[
\text{dvi\_name} \leftarrow \text{extend\_filename(cmdline(optind), ‘dvi’);} \\
\text{end; }
\]
This code is used in section 3*. 

---

\(*\) While sections 3 and 112 are mentioned in the text, it's not clear if there are specific changes or differences between them in the context of this code. The comment notes that section 3* is used, suggesting there might be a reference to section 3 that is not directly visible in the visible text. The asterisk indicates a footnote or reference that is not shown in the provided content.
§113*   Here are the options we allow. The first is one of the standard GNU options.

\begin{verbatim}
(Define the option table 113*) \equiv
  current_option \leftarrow 0; long_options[current_option].name \leftarrow "help";
  long_options[current_option].has_arg \leftarrow 0; long_options[current_option].flag \leftarrow 0;
  long_options[current_option].val \leftarrow 0; incr(current_option);
\end{verbatim}

See also sections 114*, 115*, 116*, 118*, 119*, 120*, 121*, and 123*.

This code is used in section 112*.

§114*   Another of the standard options.

\begin{verbatim}
(Define the option table 113*) \equiv
  long_options[current_option].name \leftarrow "version";
  long_options[current_option].has_arg \leftarrow 0;
  long_options[current_option].flag \leftarrow 0;
  long_options[current_option].val \leftarrow 0; incr(current_option);
\end{verbatim}

§115*   How verbose to be.

\begin{verbatim}
(Define the option table 113*) \equiv
  long_options[current_option].name \leftarrow "output-level";
  long_options[current_option].has_arg \leftarrow 1;
  long_options[current_option].flag \leftarrow 0;
  long_options[current_option].val \leftarrow 0; incr(current_option);
  out_mode \leftarrow the_works;  \{ default \}
\end{verbatim}

§116*   What page to start at.

\begin{verbatim}
(Define the option table 113*) \equiv
  long_options[current_option].name \leftarrow "page-start";
  long_options[current_option].has_arg \leftarrow 1;
  long_options[current_option].flag \leftarrow 0;
  long_options[current_option].val \leftarrow 0; incr(current_option);
\end{verbatim}

§117*   Parsing the starting page specification is a bit complicated.

\begin{verbatim}
(Determine the desired start_count values from optarg 117*) \equiv
  k \leftarrow 0;  \{ which \ count register we're on \}
  m \leftarrow 0;  \{ position in optarg \}
  \textbf{while} optarg[m] \textbf{do}
  \textbf{begin if} optarg[m] = "*" \textbf{then}
    \textbf{begin start_there[k] \leftarrow false; incr(m);}
  \textbf{end else if} optarg[m] = "." \textbf{then}
    \textbf{begin incr(k);}
      \textbf{if} k \geq 10 \textbf{then}
        \textbf{begin write Ln(stderr, my_name, ' \\ More than \ count registers specified.');}
          uexit(1);
        \textbf{end;}
        incr(m);
      \textbf{end else begin} start_count[k] \leftarrow strtol(optarg + m, address_of(end_num), 10);
      \textbf{if} end_num = optarg + m \textbf{then}
        \textbf{begin write Ln(stderr, my_name, ' \ -page-start values must be numeric or *.');}
          uexit(1);
        \textbf{end;}
        start_there[k] \leftarrow true; m \leftarrow m + end_num - (optarg + m);
      \textbf{end;}
      start_vals \leftarrow k;
  \textbf{end;}
\end{verbatim}

This code is used in section 112*.
118* How many pages to do.
\(\langle\text{Define the option table 113}\rangle\) +≡
\long_options[\text{current}].name \leftarrow \text{`max-pages`}; 
\long_options[\text{current}].has_arg \leftarrow 1;
\long_options[\text{current}].flag \leftarrow 0; \long_options[\text{current}].val \leftarrow 0; \text{incr(\text{current})};
\text{max\_pages} \leftarrow 1000000; \{\text{default}\}

119* Resolution, in pixels per inch.
\(\langle\text{Define the option table 113}\rangle\) +≡
\long_options[\text{current}].name \leftarrow \text{`dpi`}; \long_options[\text{current}].has_arg \leftarrow 1;
\long_options[\text{current}].flag \leftarrow 0; \long_options[\text{current}].val \leftarrow 0; \text{incr(\text{current})};
\text{resolution} \leftarrow 300.0; \{\text{default}\}

120* Magnification to apply.
\(\langle\text{Define the option table 113}\rangle\) +≡
\long_options[\text{current}].name \leftarrow \text{`magnification`}; \long_options[\text{current}].has_arg \leftarrow 1;
\long_options[\text{current}].flag \leftarrow 0; \long_options[\text{current}].val \leftarrow 0; \text{incr(\text{current})};
\text{new\_mag} \leftarrow 0; \{\text{default is to keep the old one}\}

121* Whether to show numeric opcodes.
\(\langle\text{Define the option table 113}\rangle\) +≡
\long_options[\text{current}].name \leftarrow \text{`show-opcodes`}; \long_options[\text{current}].has_arg \leftarrow 0;
\long_options[\text{current}].flag \leftarrow \text{address_of(show\_opcodes)}; \long_options[\text{current}].val \leftarrow 1;
\text{incr(\text{current})};

122* (Globals in the outer block 10) +≡
\text{show\_opcodes: c_int\_type};

123* An element with all zeros always ends the list.
\(\langle\text{Define the option table 113}\rangle\) +≡
\long_options[\text{current}].name \leftarrow 0; \long_options[\text{current}].has_arg \leftarrow 0;
\long_options[\text{current}].flag \leftarrow 0; \long_options[\text{current}].val \leftarrow 0;

124* Global filenames.
\(\langle\text{Globals in the outer block 10}\rangle\) +≡
\text{dvi\_name: const c\_string};
125* Index. Pointers to error messages appear here together with the section numbers where each identifier is used.

The following sections were changed by the change file: 1, 3, 4, 5, 7, 8, 9, 23, 24, 28, 42, 43, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 59, 62, 64, 65, 66, 75, 80, 99, 107, 110, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125.

-dpi: 119*
-help: 113*
-magnification: 120*
-max-pages: 118*
-output-level: 115*
-page-start: 116*
-show-opcodes: 121*
-version: 114*

b2: 25, 26, 35, 37.
b3: 25, 26, 35, 37.
c: 27, 59*
c_int_type: 112* 122*
change_font: 77, 82, 86.
char: 9* 23* 24* 60* 112*
char_pixel_width: 39, 89.
char_width: 30, 39, 89.
char_width_end: 30, 39.

character c invalid...: 89.
check sum: 18.
check sum doesn't match: 60.
check sums do not agree: 63.
Chinese characters: 15, 89.
chr: 9* 10, 12.
cmdline: 112*
count: 42* 44, 99*, 102, 111.
cur_font: 77, 78, 79, 84, 85, 89, 94.
cur_name: 23* 24* 62* 66*.
cur_pos: 28*.
current_option: 112* 113*, 114*, 115*, 116*, 118*.

d: 27, 59*
decr: 6, 83, 96, 100, 109, 111.
deeper than claimed...: 83.
default_directory: 64*.
den: 15, 17, 19.
denominator: 39, 103, 110*.
denominator doesn't match: 103.
denominator is wrong: 110*.
design size doesn't match: 60.
design sizes do not agree: 63.
do nothing: 6, 96, 112*.
do_page: 71, 75*, 77, 78, 79, 81, 83, 95, 111.
done: 4* 79, 80*, 81, 82, 83, 86, 87, 89, 90, 91, 92, 93, 94, 111.
down_the_drain: 95, 96.
down1: 15, 16, 75*, 85.
down2: 15.
down3: 15.
down4: 15.
DVI files: 13.
font name doesn't match: 60.

four_cases: 75* 81, 82, 84, 85, 86, 96.

free: 62*

Fuchs, David Raymond: 1* 13, 20.

full_name: 23*

get_byte: 27, 28* 61, 75* 80* 87, 96, 99* 100, 102, 105, 106, 109.

get_three_bytes: 27, 75*.

get_two_bytes: 27, 75*, 103.

getopt: 112*

godopt_long_only: 112*

godopt_return_val: 112*

godopt_struct: 112*

h: 72.


hh: 72, 79, 83, 84, 89, 90, 91, 93.

hhh: 79, 91.

hhstack: 72, 83.

hstack: 72, 83.

i: 2* 17.

ID byte is wrong: 100.

id_byte: 17, 100, 105, 109.

identification...should be n: 105, 109.

illegal command at byte n: 96.

in_postamble: 57, 58, 59*, 95, 99* 102, 107*, 111.

in_TFM: 34, 37, 62*.

in_width: 33, 37, 40.


infinity: 91, 92.

initialize: 2* 107*.

input_map: 47*.


invalid_font: 30, 31, 32, 79, 89, 94.

invalid_width: 30, 40, 89.

j: 59*.

Japanese characters: 15, 89.

jump_out: 7*.

k: 17, 32, 34, 44, 59*, 69, 82, 95, 99* 103, 108.

kpse_find_tfm: 23* 62*.

ekpse_set_program_name: 3*.

last_text_char: 9* 12.

ih: 34, 35.

line_length: 5* 67, 69, 70.


m: 59*, 103, 108.

mag: 15, 17, 18, 19, 39, 103, 110*.

magnification doesn't match: 103.

magnification is wrong: 110*.
observed maxh was x: 104.
observed maxstackdepth was x: 104.
observed maxv was x: 104.
old_backpointer: 97, 98, 99, 102, 107.*
only n bytes long: 100.
open_tfm_file: 23* 24* 62*.
optarg: 112* 117*.
optind: 112*.
option_index: 112*.
Options selected: 56*.
ord: 10.
right1: 15, 16, 75, 84.
right2: 15.
right3: 15.
right4: 15.
round: 35, 40, 61, 63.
rule_pixels: 15, 76, 90.
s: 78.
scaled: 61.
scaled size doesn't match: 60.
scan_hop: 95, 99*111.
set_char_0: 15, 16, 75, 81.
set_char_1: 15.
set_char_127: 15.
set_pos: 28*.
set_rule: 13, 15, 16, 75, 81, 96.
set1: 15, 16, 75, 81.
set2: 15.
set3: 15.
set4: 15.
show: 80*.
show_opcodes: 80*121*122*.
show_state: 77, 79, 80, 83.
showing: 61, 78, 80*87, 90, 91, 92, 93, 94, 95, 103.
signature...should be...: 105.
signed_byte: 27, 75*.
signed_pair: 27, 75*.
signed_quad: 27, 61, 75*, 90, 96, 99*, 100, 102, 103, 105, 107*, 110*.
signed_trio: 27, 75*.
sixteen_cases: 75*.
sixty_four_cases: 75*, 86.
skip_pages: 95, 107*.
sp: 17.
special_cases: 78, 81, 82.
ss: 78, 83, 93.
stack not empty...: 83.
stack_size: 7*26, 74, 83.
start_count: 42*, 44, 56*, 117*.
start_loc: 101, 102.
start_match: 44, 95, 102.
start_space: 42*, 44, 56*, 117*.
start_typs: 42*, 44, 56*, 111, 117*.
started: 95, 97, 98.
starting page number...: 102.
stderr: 7*, 112*, 117*.
stdout: 3*.
strcmp: 112*.
string of negative length: 87.
strncpy: 66*.
strtol: 117*.
system dependencies: 2, 7*, 9*, 20, 21, 23*, 26, 27, 28*, 40, 41, 46, 47*, 64*, 66*.
term_out: 46.
terse: 41, 50*, 80*.
text_buf: 67, 69, 70.
text_char: 9*10.
text_file: 9*.
text_ptr: 67, 68, 69, 70.
TFM files: 29.
TFM file can't be opened: 62*.
TFM file is bad: 34.
tfm_check_sum: 33, 35, 63.
tfm_conv: 33, 35, 110*.
tfm_design_size: 33, 35, 63.
tfm_file: 22, 23*, 26, 33, 35, 62*.
the file ended prematurely: 80*, 96, 99*.
the_works: 41, 56*, 57, 59*, 100, 103, 107*, 115*.
there are really n pages: 102, 104.
thirty_two_cases: 75*.
this font is magnified: 63.
this font was already defined: 59*.
this font wasn't loaded before: 59*.
total_pages: 73, 102, 103, 104.
true: 2, 28*, 34, 42*, 44, 60, 79, 80*, 82, 83, 87, 95, 99*, 100, 102, 107*, 117*.
true_conv: 39, 61, 63, 110*.
trunc: 76.
uxit: 7*, 112*, 117*.
UNDEFINED: 32.
undefined command: 82.
undefined_commands: 16, 75*, 96.
update_terminal: 46.
usage: 112*.
usage_help: 112*.
v: 72.
verbose: 41, 56*, 80*.
version_string: 3*.
vv_stack: 72, 83.
vv: 72, 79, 83, 85, 92, 93.
vv_stack: 72, 83.
vv: 82, 92.
w: 72.
warning: |h|...: 91.
warning: |v|...: 92.
warning: observed maxh...: 104.
warning: observed maxstack...: 104.
warning: observed maxv...: 104.
width: 30, 36, 39, 40.
width_base: 30, 39, 40.
width_ptr: 30, 31, 34, 35, 36, 40.
w: 34, 35, 36, 40.
write: 3*.
write_in: 3*, 7*, 112*, 117*. 
wstack: 72, 83.
w0: 15, 16, 75, 84.
w1: 15, 16, 75, 84.
w2: 15.
w3: 15.
w4: 15.
x: 17, 72.
xchr: 10, 11, 12, 32, 66, 69, 87, 109.
xfclose: 62*
xfseek: 28*
xftell: 28*
xmalloc: 62*
xmalloc_array: 66*
xord: 10, 12.
xstack: 72, 83.
xxx1: 15, 16, 75, 82, 96.
xxx2: 15.
xxx3: 15.
xxx4: 15, 16.
x0: 15, 16, 75, 84.
x1: 15, 16, 75, 84.
x2: 15.
x3: 15.
x4: 15.
y: 72.
ystack: 72, 83.
y0: 15, 16, 75, 85.
y1: 15, 16, 75, 85.
y2: 15.
y3: 15.
y4: 15.
z: 34, 72.
zstack: 72, 83.
z0: 15, 16, 75, 85.
z1: 15, 16, 75, 85.
z2: 15.
z3: 15.
z4: 15.
Names of the sections

DVI type changes for C

(Cases for commands nop, bop, ..., pop 83) Used in section 81.
(Cases for fonts 86) Used in section 82.
(Cases for horizontal motion 84) Used in section 81.
(Cases for vertical motion 85) Used in section 82.
(Classes that the current font definition matches the old one 60) Used in section 59.*
(Compare the last parameters with the accumulated facts 104) Used in section 103.
(Compute the conversion factors 110*) Used in section 109.
(Counts in the outer block 5*) Used in section 3*
(Count the pages and move to the starting page 102) Used in section 107*
(Declare the function called special_cases 82) Used in section 79.
(Declare the procedure called scan_bop 99*) Used in section 95.
(Define the option table 113*, 114*, 115*, 116*, 118*, 119*, 120*, 121*, 123*) Used in section 112*
(Define parse_arguments 112*) Used in section 3*
(Determine the desired start_count values from optarg 117*) Used in section 112*
(Find the postamble, working back from the end 100) Used in section 107*
(Finish a command that changes the current font, then goto done 94) Used in section 82.
(Finish a command that either sets or puts a character, then goto move_right or done 89) Used in section 80*
(Finish a command that either sets or puts a rule, then goto move_right or done 90) Used in section 80*
(Finish a command that sets h ← h + q, then goto done 91) Used in section 80*
(Finish a command that sets v ← v + p, then goto done 92) Used in section 82.
(Finish loading the new font info 63) Used in section 62*
(Globals in the outer block 10, 22, 24*, 25, 30, 33, 39, 41, 42*, 57, 67, 72, 73, 78, 97, 101, 108, 122*, 124*) Used in section 3*
(Lables in the outer block 4*) Used in section 3*
(Labels of the fonts 62*) Used in section 59*.
(Make sure that the end of the file is well-formed 105) Used in section 103.
(Move font name into the cur_name string 66) Used in section 62*.
(Move the widths from in_width to width, and append pixel_width values 40) Used in section 34.
(Print all the selected options 56) Used in section 107*.
(Process the font definitions of the postamble 106) Used in section 103.
(Process the preamble 109) Used in section 107*.
(Read and convert the width values, setting up the in_width table 37) Used in section 34.
(Read past the header data; goto 9997 if there is a problem 35) Used in section 34.
(Read the font parameters into position for font nf, and print the font name 61) Used in section 59*.
(Replace z by z′ and compute α, β 38) Used in section 37.
(Set initial values 11, 12, 31, 58, 68, 74, 98) Used in section 3*
(Show the values of ss, h, v, w, x, y, z, hh, and vv; then goto done 93) Used in section 80*.
(Skip until finding eop 96) Used in section 95.
(Start translation of command o and goto the appropriate label to finish the job 81) Used in section 80*.
(Store character-width indices at the end of the width table 36) Used in section 34.
(Translate a set_char command 88) Used in section 81.
(Translate an xxx command and goto done 87) Used in section 82.
(Translate the next command in the DVI file; goto 9999 with do_page = true if it was eop; goto 9998 if premature termination is needed 80*) Used in section 79.
(Translate up to max_pages pages 111) Used in section 107*.
(Types in the outer block 8*, 9*, 21) Used in section 3*.